UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. Ingenieros Informáticos

Trabajo fin de Máster

Máster en Ingeniería Informática

Adaptación de una herramienta de Generación de Lenguaje Natural al idioma Español

Autor: Damian Jozef Trzpis

Director: Martín Molina González

MADRID, 05 DE JULIO DE 2015

Agradecimientos

Después de dos años de arduo trabajo, esfuerzo, compromiso, y dedicación, el Presente Trabajo de Fin de Máster es la representación física de otra meta cumplida, en lo que al ámbito profesional se refiere. Es por ello que me gustaría agradecer a todas aquellas personas que han servido de gran ayuda y apoyo durante todo este tiempo.

A mi familia, amigos, compañeros de Máster (¡viva el delegado!), y novia, gracias por su apoyo y paciencia.

A mi director de tesis Martin Molina, gracias por el conocimiento compartido y las directrices dadas que han servido de ayuda para el desarrollo de este Proyecto.

Resumen

En el presente Trabajo de Fin de Máster se ha realizado un análisis sobre las técnicas y herramientas de Generación de Lenguaje Natural (GLN), así como las modificaciones a la herramienta *Simple NLG* para generar expresiones en el idioma Español.

Dicha extensión va a permitir ampliar el grupo de personas a las cuales se les transmite la información, ya que alrededor de 540 millones de personas hablan español.

Keywords - Generación de Lenguaje Natural, técnicas de GLN, herramientas de GLN, Inteligencia Artificial, análisis, SimpleNLG.

Abstract

In this Master's Thesis has been performed an analysis on techniques and tools for Natural Language Generation (NLG), also the *Simple NLG* tool has been modified in order to generate expressions in the Spanish language.

This modification will allow transmitting the information to more people; around 540 million people speak Spanish.

Keywords - Natural Language Generation, NLG tools, NLG techniques, Artificial Intelligence, analysis, SimpleNLG.

ÍNDICE

Listado de Figuras

Listado de Tablas

Introducción y Objetivos

La Inteligencia Artificial, es una de las ramas de la Ingeniería Informática, que persigue como máximo objetivo crear máquinas que puedan reproducir el Lenguaje Natural, no sólo leerlo o escribirlo sino que también interpretarlo, al igual que lo hace el ser humano.

La Generación de Lenguaje Natural, es el proceso de construcción de un texto que pueda ser utilizado para establecer una comunicación. Basándose en ésta idea, el presente Trabajo de Fin de Máster, es un proyecto de investigación y desarrollo, que se centra en el análisis de técnicas y/o herramientas que permiten la generación de Lenguaje Natural y la inclusión de una extensión a la herramienta *SimpleNLG* que permite generar frases en el idioma Español.

El proceso de análisis está dividido en dos fases:

- La primera corresponde al estudio del Lenguaje Natural, la generación del mismo y el estudio de las técnicas que son utilizadas hoy en día con tal fin.

- La segunda, está relacionada con el estudio y entendimiento de la gramática española, la cual sirve de base para el desarrollo de la extensión de la librería *SimpleNLG* que permite crear frases en español.

En cuanto al proceso de desarrollo, está comprendido por el estudio del estado actual de la librería *SimpleNLG*, es decir, el conocimiento de las clases y paquetes, de manera que sirvan como base para la codificación de la extensión.

Para el desarrollo del Trabajo de Fin de Máster se han propuesto los siguientes objetivos:

- Análisis del problema de Generación de Lenguaje Natural (a partir de ahora, GLN) herramientas software de GLN.
- Estudiar la herramienta software SimpleNLG como herramienta representativa de GLN.
- Diseñar una solución para la adaptación de SimpleNLG al idioma Español.
- Realizar la implementación de solución propuesta a nivel de prototipo.
- Validar el trabajo de adaptación realizado.

El primer objetivo hace referencia a los conceptos que se consideran necesarios clarificar para la mejor comprensión del proyecto y a trabajos han sido realizados previamente para conocer el avance o estado actual de los logros de la Inteligencia Artificial y la Generación del Lenguaje Natural.

La implementación de la extensión comprende la aplicación de las reglas lingüísticas aplicadas y un conjunto de pruebas que demuestran el funcionamiento de dicha extensión y que pueden para establecer conclusiones del trabajo realizado

Capítulo 1

GENERACIÓN DE LENGUAJE NATURAL

1. Generación de Lenguaje Natural

En esta sección se describen conceptos básicos indispensables para el correcto entendimiento del Trabajo realizado. Dichos conceptos están relacionados con la Generación de Lenguaje Natural, y el Texto (estructura, contenido, lexicalización, entre otros)

Desde la aparición de la computadora, uno de los mayores objetivos ha sido poder interactuar con la máquina de una manera más intuitiva. Está claro, que ésta interacción será más fácil o supondrá menos esfuerzo para las personas, si éstas pueden servirse de su modo habitual de comunicación, que en la mayoría de las ocasiones es el Lenguaje Natural (a partir de ahora, LN).

1.1. Procesamiento de Lenguaje Natural

El Procesamiento de Lenguaje Natural (a partir de ahora, PLN) es una disciplina de la Inteligencia Artificial, cuyo cometido es abordar las cuestiones planteadas por los sistemas informáticos que intentan entender o producir una o más lenguas humanas.

Dentro del PLN, se pueden destacar dos grandes bloques:

- Comprensión de Lenguaje Natural (a partir de ahora, CLN), enfocado a entender texto o voz.
- Generación de Lenguaje Natural (a partir de ahora, GLN), se ocupa de generar textos o voz en alguna lengua humana. Este trabajo está enfocado en la GLN.

1.2.Tareas involucradas en la GLN

Para ofrecer una mejor descripción del proceso de GLN, es necesario analizar las tareas involucradas. Dentro de la Comunidad Desarrolladora e Investigadora de GLN existe un consenso sobre la funcionalidad lingüística general de un sistema de GLN, compuesta por distintas clasificaciones.

Una de las más comúnmente aceptadas es la de Reiter y Dale [1997], depurada por ellos mismos posteriormente en [Reiter y Dale, 2000]. Otra, propuesta por Cahill y Reape [1999] en el proyecto RAGS [RAGS, 2000], también muy reconocida en la comunidad.

Pero la que se ha decidido estudiar con más detalle, es la clasificación propuesta por Mª del Socorro Bernardos [2007] en un estudio temático, *"¿Qué es la generación de lenguaje natural? Una visión general sobre el proceso de generación".* Propone que el proceso de GLN es el resultado de la realización de las siguientes tareas:

- **Determinación del contenido del texto.** Decide qué información comunicar en el texto de salida.
- **Estructuración del texto.** Determina las relaciones retóricas existentes entre los elementos informativos y establece un orden general entre estos últimos.
- **Lexicalización del texto.** Elige los elementos léxicos que se usarán para expresar la información especificada.
- **Agregación del texto.** Establece las combinaciones que se han de realizar entre los elementos informativos para dar lugar a oraciones y, por tanto, completa el orden en el que se deben expresar.

1.2.1. Determinación del contenido del texto

Es el proceso de decidir y obtener la información que se debe comunicar en un texto. Esta tarea está íntimamente relacionada con otras tareas de generación, como la estructuración y la generación de expresiones de referencia.

Para llevarla a cabo puede ser necesario tener en cuenta aspectos lingüísticos, por ejemplo, decidir qué información proporcionar sobre una entidad para que se pueda identificar claramente, sin ambigüedad. La información que se debe incluir en un texto y las circunstancias en las que se debe incluir dependen mucho de la aplicación.

Algunos de los factores de los que depende la elección del contenido son los siguientes:

- **Objetivos comunicativos**. Por ejemplo, la información para describir cómo se ha originado un tipo de fenómeno concreto, será distinta a la requerida para intentar convencer de que ese fenómeno es bueno para el usuario.
- **Características del usuario**. Por ejemplo, alguien considerado novato en el dominio puede necesitar más información explicativa que alguien considerado experto.
- **Contexto**. Por ejemplo, si ya se ha hablado de una entidad, puede no ser necesario volver a obtener la misma información sobre ella.
- **Requisitos del sistema.** Por ejemplo, puede ser necesario que el texto se ajuste a un espacio dado, por lo que no se puede incluir todo la información que se desee en un principio.

La determinación del contenido es una tarea muy dependiente de la aplicación, y esto hace que en la práctica se prefieran enfoques *ad hoc* frente a otros más generales.

1.2.2. Estructuración del texto

La tarea de estructuración del texto consiste en organizar los elementos informativos que se han de comunicar de forma que resulte un texto coherente y no una mera recopilación de elementos ordenados al azar. Esta tarea determina las relaciones retóricas que se dan entre los elementos o grupos de elementos para indicar cómo están relacionados en el discurso los fragmentos del texto (por ejemplo, contraste, elaboración, entre otros).

Si se pretende que los textos que se generen sean semántica y estructuralmente ricos y reflejen una cierta calidad, se han de establecer y aplicar criterios para elegir entre las variantes. Para esto se puede elegir la forma de estructuración siguiendo básicamente los mismos criterios que dirigen la tarea de determinación del contenido:

- **Objetivos comunicativos**. Por ejemplo, la estructura para describir cómo se ha originado un tipo de fenómeno concreto, será distinta de la requerida para comparar ese fenómeno con otro.
- **Características del usuario**. Por ejemplo, un usuario puede preferir la información de forma esquemática, mientras que otro puede desear un texto más elaborado.
- **Contexto.** Por ejemplo, cambiar la forma de organización que se ha estado siguiendo, puede resultar confuso para el usuario.
- **Requisitos del sistema**. Por ejemplo, se puede requerir una estructura especial para señalar primero los datos que tengan valores que se salgan de lo normal.

Los enfoques de la estructuración se clasifican en dos grupos:

- **Enfoques de arriba abajo**, en los que se elige la estructura que se quiere lograr y ésta se va descomponiendo hasta el nivel de los elementos informativos.
- **Enfoques de abajo arriba**, donde se parte de los elementos informativos para combinarlos hasta obtener una representación de la estructura del texto.

Los enfoques de arriba abajo, especialmente los esquemas, dan lugar a sistemas más rápidos que los de abajo arriba.
Sin embargo, con los enfoques de abajo arriba es más fácil satisfacer diversos objetivos comunicativos y es posible forzar la comunicación de unos datos dados.
Los esquemas constituyen la opción con un procesamiento más sencillo, pero mezclan la estructuración con otras tareas, como la agregación, y proporcionan menos flexibilidad que la planificación y los enfoques de abajo arriba. Por otro lado, estos dos últimos requieren recursos complejos y difíciles de construir. Como ocurre en el resto de tareas, siempre que las características del sistema lo permitan, lo más aconsejable es utilizar las técnicas más sencillas, que suelen corresponder a enfoques más superficiales.

1.2.3. Lexicalización del texto

Consiste en determinar los elementos léxicos que se necesitan para expresar los elementos informativos especificados. El tratamiento morfológico no está incluido en esta tarea. Como en el caso de las demás tareas, la complejidad de la lexicalización depende de la sofisticación del sistema de GLN.

Hay dos tipos de lexicalización:

- **Sencilla**, cuando la correspondencia entre los elementos del texto y los términos concretos de la lengua es única.
- **Elección léxica**, cuando hay que optar entre varias posibilidades.

Puede haber casos donde no hay lexicalización, ya que todas las palabras están completamente determinadas por la entrada. Por otro lado, los elementos léxicos pueden estar asociados a más de un lexema, palabra o frase.
En los casos de elección léxica, ésta no sólo está restringida por el conocimiento conceptual específico del dominio, sino que se han de considerar varios factores, entre los que se pueden señalar los siguientes:

- **Objetivos comunicativos.** Por ejemplo, la expresión explícita de una lista de elementos puede ser poco fluida y probablemente se debe evitar; sin embargo, puede resultar adecuada si se tiene como objetivo pragmático enfatizar.
- **Características del usuario**. Distintos usuarios pueden requerir una lexicalización diferente. Algunos términos técnicos que se pueden usar para usuarios expertos, pueden no ser apropiados para novatos.
- **Contexto.** Por ejemplo, cambiar la lexicalización de un concepto para que no siempre se exprese igual.
- **Requisitos del sistema.** Por ejemplo, no es lo mismo escribir un texto de género científico que uno periodístico.
- **Restricciones léxicas.** Por ejemplo, el estudio de las colocaciones de las palabras en una lengua puede servir para elegir cuál es la más adecuada para que acompañe a otra.
- **Restricciones sintácticas.** Por ejemplo, al elegir los lexemas conviene considerar los patrones que gobiernan la combinación de las palabras, como la estructura temática de los verbos.
- **Consideraciones pragmáticas.** Por ejemplo, puede ser adecuada una lexicalización diferente en circunstancias formales y en coloquiales, o en connotaciones.

Por último, conviene señalar que muchos sistemas no especifican las relaciones entre los componentes de un texto, mediante conceptos abstractos, sino que introducen las palabras conectoras o los marcadores de discurso (por ejemplo, “pero”, “aunque”, entre otras.) que los expresan, por lo que también determinan el modo en que se debe lexicalizar la relación.

1.2.4. Agregación del texto

Bernardos considera la agregación como la combinación de elementos informativos tanto dentro como entre ellos, y el ordenamiento de estos elementos y de las oraciones resultantes con el fin de conseguir un texto lo más fluido y legible posible. La combinación de elementos informativos implica el uso de recursos lingüísticos para construir unidades que comunican varios elementos informativos a la vez.

Existen distintos mecanismos para hacer agregación y, entre éstos, los más estudiados han sido los que involucran elementos expresados mediante cláusulas u oraciones; por ejemplo, la conjunción simple, la conjunción mediante elementos compartidos y la inclusión.
Un sistema de GLN debe decidir qué mecanismo de agregación usar, en caso de que considere necesario hacer alguna. En términos generales, existen dos aspectos que suelen estar en conflicto: concisión y simplicidad sintáctica. Por un lado, la agregación es un mecanismo útil para acortar los textos. Por otro lado, la agregación hace que las oraciones sean más complejas sintácticamente hablando. El tipo de agregación adecuado también depende de los siguientes aspectos:

- **Objetivos comunicativos.** Por ejemplo, se puede pensar que los textos de acción–condición se pueden comprender más rápidamente si se especifica la acción primero [Dixon, 1982], o con el fin de evitar malas interpretaciones es mejor poner primero la condición, ya que si la acción está primero, el lector puede ejecutarla inmediatamente antes de darse cuenta de que tiene una condición asociada. El orden dependerá en parte de lo importante que es minimizar el tiempo de lectura o la tasa de error.
- **Características del usuario.** Por ejemplo, el usuario puede preferir un estilo sencillo, con oraciones simples, frente a otro más elaborado.
- **Contexto.** Por ejemplo, la repetición de una explicación dada puede ser útil para que el texto sea más comprensible.
- **Consideraciones semánticas.** Por ejemplo, parece que la agregación es más aceptable cuando los elementos informativos están relacionados semánticamente.
- **Requisitos del sistema**. Por ejemplo, en casos en que el espacio sea limitado, se favorecerá la concisión.

1.2.5. Generación de las expresiones de referencia del texto.

La generación de expresiones de referencia del texto, está relacionada con la forma de producir la descripción de una entidad, de modo que el receptor pueda identificar esa entidad en un contexto dado. La descripción que se elija para hacer referencia a una entidad por primera vez (referencia inicial) dependerá de la razón por la que se introduce a esa entidad en la conversación, y de la información que se pretende transmitir posteriormente.
Si se hace referencia a la entidad después de haber aparecido en el texto (referencia posterior), la preocupación es distinguir el referente pretendido de las otras entidades con las que se podría confundir.

Mediante la generación de expresiones de referencia se reemplazan los nombres simbólicos de las entidades de la fuente de información del dominio por el contenido semántico de expresiones (generalmente, sintagmas nominales) que sean suficientes para que el receptor identifique sin ambigüedad la entidad de la que se está hablando. Se debe evitar la redundancia y la inclusión de información innecesaria en las descripciones. El contexto es el factor principal a la hora de generar expresiones de referencia, pero, como en el resto de las tareas también es importante contar con otros aspectos, como los siguientes:

- **Objetivos comunicativos**. Por ejemplo, si se pretende explicar el funcionamiento de un artefacto peligroso, puede resultar conveniente usar pocas expresiones anafóricas para eliminar cualquier ambigüedad.

- **Características del usuario.** Por ejemplo, si el receptor del texto conoce el tema que se está tratando, podrá identificar las entidades sin necesidad de emplear expresiones definidas muy elaboradas.
- **Requisitos del sistema**. Por ejemplo, la aplicación puede producir documentos que contengan imágenes. Si se desea que éstas no estén aisladas, sino que texto e imágenes se complementen, el texto generado ha de referirse a ellas en algún momento.

La generación de expresiones de referencia se ha centrado en el uso de pronombres, nombres propios y sintagmas nominales completos; no se ha abordado en profundidad el uso de otras formas de referencia como los términos deícticos y el uso de expresiones anafóricas. El mecanismo más empleado para generar referencias no ambiguas es el algoritmo incremental [Dale y Reiter, 1995], o alguna de sus variantes [Krahmer et al., 2001], puesto que, además de dar lugar a expresiones adecuadas, es muy eficiente desde el punto de vista computacional.

1.2.6. Realización gramatical del texto

La realización gramatical desempeña su labor en el nivel de la oración, a diferencia del resto de las tareas, que trabajan con una visión del texto completo o partes más o menos amplias de él, o con componentes más pequeños. Al igual que los textos no son secuencias de oraciones ordenadas al azar, las oraciones no son secuencias de palabras ordenadas al azar. Cada lengua está definida, al menos en parte, por un conjunto de reglas gramaticales que especifican lo que es una oración bien formada en esa lengua.
La realización gramatical consiste en aplicar alguna caracterización de estas reglas de la gramática a una representación más o menos abstracta para producir un texto que sea sintáctica y morfológicamente correcto. La complejidad del proceso de realización depende de la distancia entre la forma de superficie y la representación abstracta. Algunos ejemplos sencillos de los aspectos sintácticos y morfológicos que se han de tener en cuenta durante la realización gramatical son los siguientes:

- **Reglas sobre la formación de grupos verbales:** Parte de esta tarea consiste en construir un grupo verbal adecuado según el tiempo, la forma general (interrogación, orden, etc.), la polaridad (negación, afirmación), etc.
- **Reglas sobre concordancia:** Se ha de conseguir que se cumplan estas reglas mientras construye la oración. Por ejemplo, entre el verbo y el sujeto de la oración, entre el adjetivo y el nombre al que acompaña, etc.
- **Reglas sobre pronominalización sintáctica necesaria:** En algunos casos, las reglas sintácticas requieren que se usen pronombres en las oraciones. Por ejemplo, pronombres reflexivos.
- **Reglas morfológicas:** Por ejemplo, las reglas de formación del plural.

También es importante el requisito adicional de que el texto producido debe ser ortográficamente correcto. Por ejemplo, la primera palabra de una oración ha de empezar con mayúscula y se debe poner un signo de puntuación al final.

La complejidad del procesamiento llevado a cabo varía de un sistema a otro, pero todos, exceptuando los basados en plantillas, donde únicamente se han de rellenar huecos en oraciones predefinidas con los términos adecuados, tienen en común que necesitan una descripción codificada de los recursos gramaticales disponibles para la lengua.

1.2.7. Realización de la presentación del texto

La generación de un texto (escrito) se completa con la presentación de superficie de ese texto. Este enfoque plasma una distinción entre lo que a veces se llama la estructura lógica y la estructura

física. Para expresar un texto, se necesita convertir construcciones lógicas en físicas. Por ejemplo, si un fragmento de texto tiene el estado lógico de ser un párrafo, tipográficamente se pude indicar haciendo una sangría en la primera línea; o si un fragmento de texto es un elemento de una lista, se puede indicar tipográficamente mediante un topo o viñeta.
Dada la disponibilidad en la actualidad de sistemas de procesamiento de texto cuyo fin es proporcionar la conversión de estructuras lógicas en dispositivos de presentación, es suficiente que los sistemas de GLN sepan cómo convertir las representaciones lógicas que usan en los tipos de marcas requeridas como entrada de estos sistemas de procesamiento de texto.

Se pueden usar sistemas de preparación o procesamiento de texto como LaTeX y Microsoft Word, o exploradores web, como Internet Explorer y Netscape, a modo de *postprocesadores* que producirán los textos físicos que se necesitan generar con la presentación adecuada.

Cada vez más sistemas de GLN usan esos componentes externos, pero en situaciones en las que el sistema de GLN necesita producir estructuras que no permiten los sistemas de presentación disponibles el propio sistema de GLN debe responsabilizarse también de convertir las especificaciones lógicas en físicas. Además de tener en cuenta las características del procesador de texto que se vaya a utilizar, en caso de que se opte por ello, existen otros aspectos que conviene considerar:

- **Objetivos comunicativos**. Por ejemplo, si se quiere resaltar el peligro de una acción, se puede emplear texto con un formato especial, que lo distinga del resto.
- **Características del usuario.** Por ejemplo, una persona con problemas de visión puede requerir un tamaño grande de letra.
- **Contexto**. Por ejemplo, si se está continuando una lista, la numeración deberá comenzar por donde se dejó, y no empezar de nuevo.
- **Requisitos del sistema.** Por ejemplo, puede ser necesario que el texto se imprima en papel de un determinado color, lo que desaconsejará el uso de ciertos tonos en el texto.
- **Consideraciones semánticas**. Por ejemplo, para enfatizar algún dato se puede hacer que éste aparezca en cursiva.

De cualquier modo, lo que conviene destacar es que es una buena idea separar la realización de la presentación del resto del sistema, por ejemplo, encapsulándola en un módulo, para que la mayor parte del sistema de GLN trabaje con la estructura lógica del sistema y no se preocupe de los detalles de la interpretación física. Es interesante señalar, finalmente, que en aquellos casos en los que el usuario del sistema de GLN es una aplicación informática, normalmente no será necesaria la tarea de presentación del texto, o será la propia aplicación la que lleve a cabo la presentación.

Conviene señalar que a pesar de no poder garantizar que este conjunto de tareas cubra todos los problemas que se han de tratar en la GLN, la autora considera que las tareas que se describen en este trabajo son representativas, en gran parte, de las operaciones involucradas en esta área y pueden proporcionar una descomposición lo suficientemente útil para comprenderla y trabajar en ella.

También es importante tener en cuenta que, aunque en un principio se han tratado por separado, generalmente estas tareas no son independientes entre sí, sino que han de interactuar, a veces de manera bastante compleja, para conseguir generar el texto deseado.

No hay acuerdo tampoco en la comunidad de GLN sobre cómo se deben estructurar dentro del sistema de GLN. En cada proyecto habrá que analizar cuáles de las relaciones existentes entre las distintas tareas son relevantes. Una vez establecidas, se podrá determinar cuál debe ser la arquitectura del sistema, en qué orden debe llevar a cabo las tareas el sistema y cómo se han de intercalar. Conviene señalar que aunque se hable de cada una de las tareas como un todo, los

distintos aspectos involucrados en cada una de ellas no han de situarse obligatoria y necesariamente en un único módulo, sino que se pueden distribuir entre distintos componentes, que, a su vez pueden realizar labores correspondientes a más de una tarea (completa o no). En los sistemas con una arquitectura más sencilla se podrá conseguir establecer una organización tal que permita ordenar los módulos secuencialmente, pero en casos más complejos, la alternativa que se puede presentar es que los componentes necesiten actuar de manera intercalada.

Por último, cabe destacar que dado el actual estado de la GLN, apenas es posible construir sistemas de generación de texto donde cada tarea de generación esté completamente guiada por principios lingüísticos, ya que existen tareas, como la determinación del contenido y la lexicalización para las que, por su naturaleza dependiente del dominio, resulta muy difícil aplicar enfoques generales. Sin embargo, otras tareas, como la realización gramatical, sí permiten emplear una perspectiva de este tipo. En consecuencia, la mayoría de los sistemas de generación se pueden caracterizar como sistemas híbridos, en el sentido de que algunas tareas de generación se llevan a cabo siguiendo lo que se conoce como un enfoque profundo (general), mientras que otras tareas se realizan usando uno superficial (más o menos a medida).

Capítulo 2

SISTEMAS DE GENERACIÓN DE LENGUAJE NATURAL

2. Sistemas de Generación de Lenguaje Natural

En esta sección se describen algunos de los Sistemas de Generación de Lenguaje Natural que son de más común uso en la actualidad. La fuente de la información se ha obtenido de la Association for Computational Linguistics [ACL, 2013].

2.1. ASTROGEN

Es un Generador de Lenguaje Natural escrito en el lenguaje Prolog. Se espera que sea utilizado por cualquier individuo. ASTROGEN se ha utilizado para la Generación de Lenguaje Natural (idioma inglés) a partir de especificaciones formales y especificaciones de STEP/EXPRESS.

ASTROGEN consta básicamente de dos módulos; un generador de profundidad y otro de superficie. (*ver Figura 1. Arquitectura de ASTROGEN*)

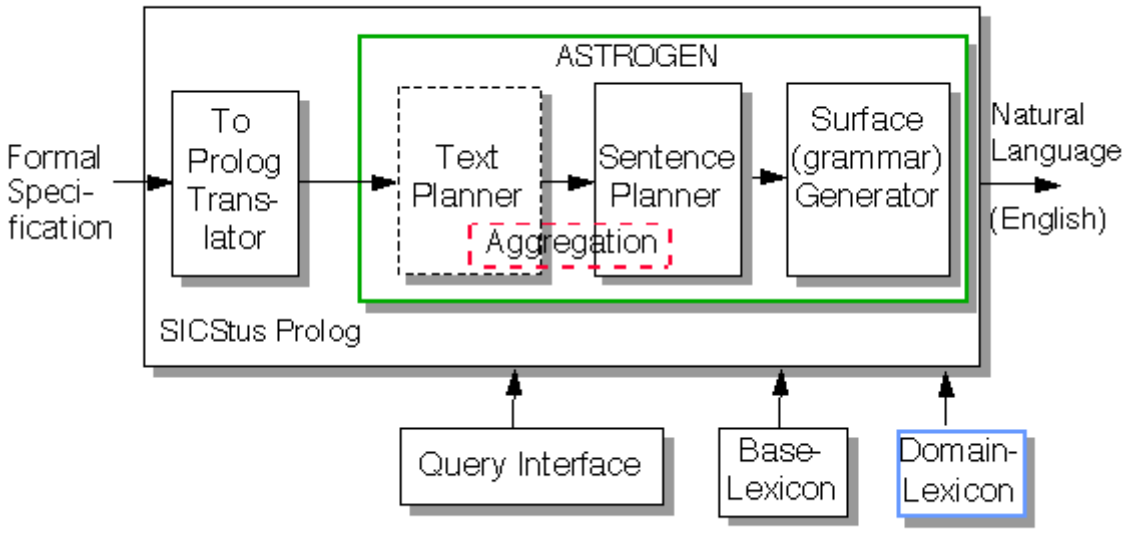

Figura 1. Arquitectura de ASTROGEN.

2.2. CLINT

Es un sistema de generación híbrido basado en plantilla/palabra con una aplicación de ejemplo de generación de una carta comercial. El sistema está escrito en C ++ y se ejecuta bajo Microsoft Windows. CLINT ha sido desarrollado por Rinat Gedalia y Michael Elhadad.

2.3. KPML

El sistema KPML ofrece una plataforma madura y robusta para la ingeniería de gramática a gran escala, que se orienta sobre todo al desarrollo de la gramática y la generación multilingüe. Esta particularmente dirigido a proporcionar recursos para aplicaciones de generación realista, pero de amplia cobertura, en los que tanto la flexibilidad de expresión y la velocidad de generación son importantes - por ejemplo en las páginas web.

KPML también se usa ampliamente en la investigación de generación de texto en varios idiomas y para la enseñanza. Se basa en la lingüística sistémica funcional.

Este sistema, es un descendiente directo del sistema de generación de texto Penman, desarrollado en forma multilingüe entre el proyecto *Komet* en Darmstadt y el Grupo Modelado Sistemático de la Universidad Macquarie.

Está disponible para Windows y Unix. El código fuente está escrito en *ANSI Common Lisp* y utiliza la interfaz *Gestor Common Lisp* (CLIM).

Un conjunto cada vez mayor de las gramáticas de generación están en desarrollo para una variedad de idiomas, incluyendo inglés, español, holandés, chino, alemán, checo entre otros.

2.4. Multimodal Unification Grammar

Multimodal Unification Grammar (MUG) Workbench es una herramienta de desarrollo y depuración para Multimodal NLG. El sistema funciona con gramáticas MUG con especificaciones de entrada arbitrarias para producir una salida en un lenguaje natural. Está diseñado para hacer tres cosas:

- *Multimodal Fission* (distribución de la salida a modos de interacción/comunicación)
- Planificación de sentencias (selección de la información para incluir en el enunciado).
- Realización gráfica de la interfaz de usuario.

El sistema MUG hace estos tres trabajos en paralelo. MUG Workbench es útil para la depuración de sus gramáticas, puede servir para inspeccionar las estructuras de datos utilizadas durante la generación.

Además, puede servir para aprender más acerca de la naturaleza de las gramáticas de unificación utilizados para el macheado o la Generación de Lenguaje Natural.

2.5. NLGen

El sistema de Generación de Lenguaje Natural NLGen aplica la estrategia *SegSim* para generar oraciones en inglés. La inferencia probabilística para la construcción de oraciones se basa en un análisis estadístico de la producción *RelEx*.

No debe confundirse con NLGen2 que utiliza una teoría de generación de frases diferente. Desarrollado en Java, bajo licencia de Apache.

2.6. NLGen2

El sistema de Generación de Lenguaje Natural NLGen2 utiliza dependencias *RelEx*, junto con el análisis de ligamiento de *Link Grammar* para generar sentencias en inglés.

No debe confundirse con NLGen, que utiliza una teoría generación frase diferente aunque también como su hermano mayor está desarrollado en Java, bajo la licencia Apache.

2.7. SimpleNLG

SimpleNLG es una API de Java sencilla diseñada para facilitar la Generación de Lenguaje Natural. Originalmente fue desarrollado en la Universidad de Aberdeen en el Departamento de Ciencias de la Computación.

SimpleNLG pretende funcionar como un "motor de realización" para arquitecturas de Generación de Lenguaje Natural, y se ha utilizado con éxito en una serie de proyectos, tanto académicos como comerciales. Se compone de las siguientes piezas:

- Léxico/morfología: El léxico predeterminado computa formas flexionadas (realización morfológica). Se considera que tiene una cobertura adecuada. Mejor cobertura se puede conseguir mediante el uso de *NIH Specialist Lexicon*).
- *Realiser*: Genera textos usando formas sintácticas. Cobertura gramatical está limitada en comparación con herramientas como KPML y FUF/SURGE, pero parece ser suficiente para muchas tareas de GLN.
- Microplanificación: Actualmente sólo agregación, en desarrollo.

Es una herramienta potente, pero a su vez simple de manejar, además está siendo usada en la Escuela Técnica Superior de Ingenieros Informáticos de la UPM, por lo tanto, fue la opción que se ha seleccionado para implementar la extensión que permita generar textos en español.

2.8. FUF/SURGE

Desarrollado por Michael Elhadad, es un intérprete escrito en *CommonLisp* para la unificación funcional basado en lenguaje, específicamente diseñado para desarrollar aplicaciones de generación de texto.

SURGE (desarrollado por Michael Elhadad y Jacques Robin) es una gramática para la generación de Inglés escrito en FUF.

Se pueden distinguir varias versiones de SURGE:

- **SURGE**
- **SURGE 2.3:** La última versión de SURGE. Incluye el soporte para diálogo por escrito, y la cobertura sintáctica ampliada basada en Penn TreeBank.
- **SURG-SP:** *Systemic Unification Reusable Grammar for Spanish* es una gramática española a gran escala que permite a sistemas que ya utilizan FUF/SURGE para GLN inglés, generar sintácticamente (y muchas veces semánticamente) texto equivalente en español. SURG-SP trabaja casi de la misma manera que SURGE 2.3.

Capítulo 3

LA HERRAMIENTA SOFTWARE SIMPLENLG

3. La herramienta software SimpleNLG

SimpleNLG es una biblioteca Java que proporciona la interfaz para el control directo sobre el proceso de "realización", es decir, sobre la forma en que se construyen y combinan las frases, sobre las operaciones morfológicas y linealización[1]. Define un conjunto léxico cubriendo la mayoría de las categorías gramaticales, así como formas de combinarlas y establece valores de diferentes características [Gatt y Reiter, 2009].

En la construcción de una estructura sintáctica y su linealización como texto, se llevan a cabo los siguientes pasos:

1. Inicializar los componentes básicos necesarios, con los elementos léxicos adecuados.
2. Utilizar las operaciones proporcionadas por la API, para inicializar características necesarias.
3. Combinar los componentes en estructuras más grandes, utilizando de nuevo, las operaciones previstas por la API.
4. Pasar la estructura resultante a la linealización, que la convierte, aplicando las inflexiones correspondientes, y la ordena, en función de las características, antes de devolverla como el texto.

SimpleNLG, además, es capaz de retornar diferentes representaciones de salida. Esto es útil en aplicaciones en las que ciertas entradas pueden ser asignadas a una cadena de salida de modo determinista, mientras que otras requieren una asignación más flexible dependiendo, por ejemplo, de las características semánticas o contexto. *SimpleNLG* intenta satisfacer estas necesidades, proporcionando cobertura sintáctica significativa.

Otro objetivo del motor es la robustez: estructuras que son incompletos o mal formadas no van a provocar fallos, pero por lo general producirán salidas poco comprensible.

Un tercer criterio de diseño fue lograr una clara separación entre operaciones morfológicas y sintácticas. El componente léxico de la biblioteca, que ofrece un generador con amplia cobertura morfológica, es distinto del componente sintáctico. Esto hace que sea útil para aplicaciones que no requieren operaciones sintácticas complejas, pero que necesitan cadenas de salida correctamente flexionadas.

3.1. GLN y SimpleNLG

¿Cuánto de GLN tiene *SimpleNLG*? La GLN tiene como objetivo producir texto comprensible, típicamente de alguna representación no lingüística de la información.

[1] Se entiende como linealización a la acción de hacer un texto coherente y conectado.

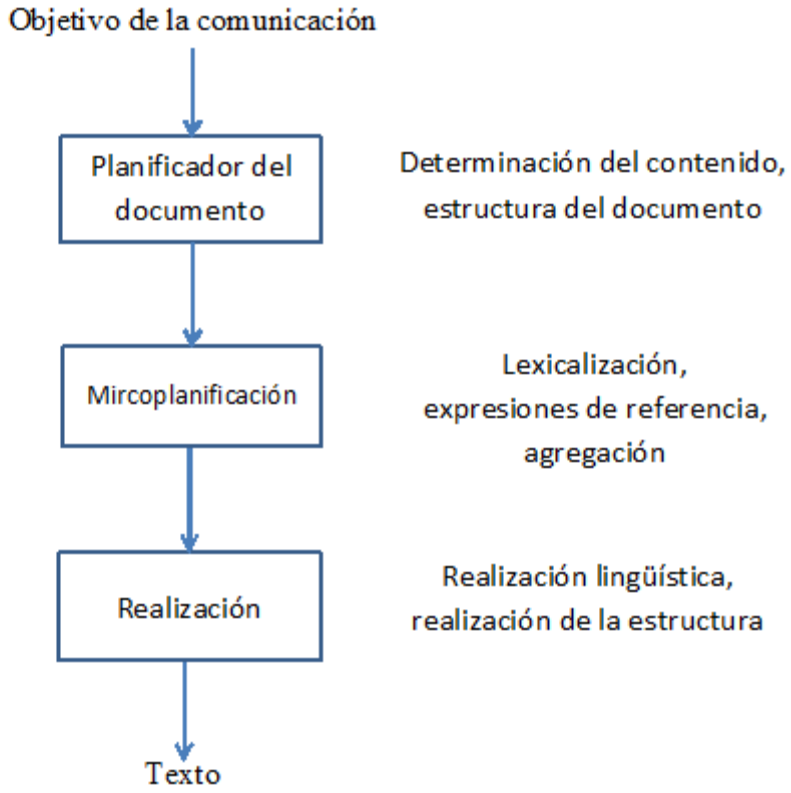

Figura 2. Sistema GLN definido por Raiter y Dale.

Muchos sistemas de GLN constan de 3 componentes que están conectados, es decir, la salida de la planificación documento actúa como entrada a la microplanificación y su salida es la entrada a la "realización". (*ver figura 2. Sistema GLN definido por Raiter y Dale*)

Planificador del documento

- **Determinación del contenido:** decide qué información aparecerá en el texto de salida. Esto depende del objetivo, el público, la información de entrada disponible y otras limitaciones, como la longitud del texto permitida.
- **Estructura del documento:** decide cómo deben agruparse los trozos de contenido en un documento, cómo relacionar estos grupos entre sí y en qué orden deberían aparecer. Por ejemplo, al describir el clima del mes pasado, es posible que hablar primero sobre la temperatura, y después de la lluvia. O, se puede comenzar hablando sobre el tiempo en general y luego proporcionar los fenómenos meteorológicos específicos que ocurrieron durante el mes.

Microplanificación

- **Lexicalización:** decide qué palabras específicas deben utilizarse para expresar el contenido. Por ejemplo, los nombres, verbos, adjetivos y adverbios para aparecer en el texto son elegidos del lexicón. Estructuras sintácticas particulares también se eligen. Por ejemplo, se puede decir "el coche propiedad de María" o 'coche de María'.
- **Expresiones de referencia:** decide qué expresiones se deben utilizar para hacer referencia a las entidades (tanto concretas, como abstractas). La misma entidad puede ser denominada de muchas maneras. Por ejemplo, marzo del año pasado puede ser denominado:

- 03/2014
- Marzo
- Marzo del año anterior

- **Agregación de expresione**s: decide cómo las estructuras creadas por el planificador del documento deben ser mapeadas en las estructuras lingüísticas tales como oraciones y párrafos. Por ejemplo, dos ideas se pueden expresar en dos oraciones: "El mes fue más frío de lo normal. El mes fue más seco de lo normal.", o con una: "El mes fue más frío y más seco de lo normal.".

Realización

- **Realización lingüística:** utiliza reglas de la gramática (de la morfología y de la sintaxis) para convertir las representaciones abstractas de frases en un texto real.
- **Realización de la estructura:** convierte estructuras abstractas tales como párrafos y frases en símbolos de marcado que se utilizan para mostrar el texto.

3.2. Generación de textos con SimpleNLG

Como se ha comentado anteriormente, *SimpleNLG* es una librería Java, motivo por el cual no tiene una clase principal con el método Main. Es un fichero "*.jar*" que tiene que ser importado en el proyecto en el que se ha de utilizar (*ver Anexo A*).

Una vez importado *SimpleNLG* al entorno de *Eclipse*, y preparado la clase Main se puede empezar a escribir código de la aplicación utilizando la API que proporciona *SimpleNLG*.

Existen diferentes opciones de crear frases, desde las más simples y menos formales, hasta las muy elaboradas, asignando meticulosamente características a cada elemento.

3.2.1. Lexicón

Al igual que otros sistemas de Procesamiento de Lenguaje Natural (PLN), *SimpleNLG* necesita conocer información acerca de las palabras con las que trabaja. Ésta información está incluida en un fichero denominado Lexicón. Por defecto, *SimpleNLG* tiene un lexicón sencillo integrado en el sistema (idioma inglés), al que se invoca de la siguiente manera:

```
Lexicon léxico = Lexicon.getDefaultLexicon ();
```

El léxico predeterminado es de 700KB, aunque también se puede utilizar el léxico 300MB NIH Especialista, que tiene una cobertura excepcional de la terminología médica.

Lo que es muy importante, y sobre todo para este Proyecto, es la posibilidad de definir un lexicón propio. Por ejemplo, si el lexicón se llama "mi lexicon.xml", y se guarda en el directorio de trabajo actual, se puede acceder de la siguiente manera:

```
Lexicon léxico = new XMLLexicon ("mi-lexicon.xml");
```

Si se encuentra fuera del directorio de trabajo hay que proporcionar la ruta completa.

Una vez incorporado el lexicón, se puede crear un objeto *NLGFactory*, el cual de fine las de *simpleNLG* y un objeto *Realiser* que transforma las estructuras *simpleNLG* en texto.

3.2.2. Generación de una frase sencilla.

El tipo más sencillo de sentencia permitida en *SimpleNLG* es una cadena de salida igual a la de entrada. Por ejemplo, en un programa que genera párrafos se quiere que siempre la primera línea del párrafo a sea "*My father builds the house*" porque todos deberían saberlo.

El código *SimpleNLG* para conseguirlo será el siguiente:

```
NLGElement s1 = nlgFactory.createSentence ("My father builds the house");
```

Ahora hay que utilizar la realización para obtener la cadena de salida:

```
String salida = realiser.realiseSentence (s1);
System.out.println (salida);
```

El resultado obtenido es el siguiente:

```
My father builds the house.
```

Es importante destacar que sólo es necesario crear una única vez el Lexicon, y los objetos *NLGFactory* y *Realiser* dentro del programa; no es necesario volver crearlos para cada frase que se genera. De modo que es una buena práctica crearlos al inicio del programa y usarlos durante la vida de la aplicación.

3.2.3. Generación de una frase más elaborada.

En el ejemplo anterior se muestra la manera más sencilla de crear una sentencia, pero esto requiere que el usuario haga la mayor parte del trabajo. Ahora se va a ver cómo conseguir que *SimpleNLG* se ocupe de ello. Hay que tener en cuenta que *SimpleNLG* es bastante flexible en este aspecto, y existen diferentes maneras de generar las mismas frases.

SimpleNLG es capaz de entender lo que es una frase utilizando la clase llamada *SPhraseSpec*. Ésta es accesible a través de *NLGFactory*, utilizando el método *createClause*.

Los conceptos de:

- Sintagma verbal
- Sintagma nominal
- Sintagma preposicional
- Sintagma adjetival
- Sintagma adverbial

También son accesibles a través de *NLGFactory*, utilizando los métodos *createVerbPhrase*, *createNounPhrase*, *createPrepositionPhrase*, entre otros. Devuelven tipos con nombres similares - *VPPhraseSpec, NPPhraseSpec, PPPhraseSpec, AdjPhraseSpec,* y *AdvPhraseSpec.*

Con el fin de construir una oración utilizando estos conceptos, hay que seguir los siguientes pasos:

- Crear una instancia de *NLGFactory*.
- Crear un contenedor para la sentencia utilizando el método *createClause* de *NLGFactory* (Devuelve una instancia de *SPhraseSpec*).
- Crear un verbo y un sustantivo utilizando los métodos *createVerbPhrase* y *createNounPhrase* de *NLGFactory* respectivamente. (Devuelven *VPPhraseSpec* y *NPPhraseSpec*.)
- Si es necesario, se pueden crear sintagmas preposicionales, adjetivales, y/o adverbiales usando los métodos *createPrepositionPhrase*, *createAdjectivePhrase,* y *createAdverbPhrase* de *NLGFactory*. (Devuelven *PPPhraseSpec*, *AdjPhraseSpec* y *AdvPhraseSpec*.)
- Indicar qué papel desempeñarán estas diversas partes de la oración en la frase deseada. Por ejemplo, se puede especificar que un sustantivo determinado sea el sujeto de la oración, y alguno otro sea el objeto, utilizando *setSubject* y *setObject*.
- Especificar el predicado, utilizando el método *setVerb*, y el complemento (si es el caso) utilizando el método *addComplement*.
- Crear un objeto de tipo *Realiser*.
- Invocar al "realizador" para que transforme la instancia *SPhraseSpec* en una cadena sintácticamente correcta.

Ahora, se dispone de una cadena de caracteres que puede ser tratada como cualquier otra cadena de caracteres de Java. Por ejemplo, puede manipularse aún más o mostrarla por consola utilizando el método *System.out.println*.

Hay que tener en cuenta que éste es el enfoque más detallado; podría simplemente utilizarse *setSubject*, *setVerb*, entre otros., pasando cadenas simples como argumentos. A menos que se quiera formar frases más complejas, especificar que un sustantivo es un *NPPhraseSpec* o que un verbo es una *VPPhraseSpec* ni siquiera es necesario.

A continuación en la Tabla 1, se muestra un breve desglose de las principales partes del discurso que *SimpleNLG* puede manejar.

Parte de la oración	**Palabra**	**Método**
Sujeto	"John"	setSubject("John")
Verbo	"give"	setVerb("give")
Objeto	"the dog"	setObject("the dog")
Complemento indirecto	"a present"	setIndirectObject("a present")
Complemento	"in the garden"	addComplement("in the garden")
Modificador	"pretty"	addModifier("pretty")
Artículo	"the"	setDeterminer("the")

Tabla 1. Ejemplos de métodos que maneja SimpleNLG.

Para saber cómo definir y combinar las diferentes partes de la oración se toma de ejemplo la frase del ejemplo anterior "*My father builds the house*". Como se ha indicado, se hace uso de la construcción *SPhraseSpec*, que permite definir una frase o cláusula en términos de sus componentes sintácticos, sin ningún orden en particular, ya que *SimpleNLG* reunirá las partes en una estructura gramatical adecuada.

Por lo tanto, se define:

```
SPhraseSpec p = nlgFactory.createClause();
p.setSubject ("My father");
p.setVerb ("build");
p.setObject ("the house");
```

Este conjunto de llamadas a *SimpleNLG* define los componentes de la frase: un sujeto, un verbo, y un objeto de la oración. Ahora, el último paso, es usar "el realizador", que combinará estos diferentes componentes de la sentencia y devolverá un texto sintáctica y morfológicamente correcto:

```
Cadena salida2 = realiser.realiseSentence (p); // Realiser creado antes.
System.out.println (salida2);
```

La salida resultante es igual al caso anterior:

```
My father build the house.
```

En este caso, el verbo se ha tenido que conjugar para que concuerde con la persona. De esa tarea se ocupa el módulo de morfología, donde están definidas las reglas morfológicas para los casos generales - en caso particular se consulta o bien la configuración del usuario y si no está definida ninguna- el lexicón.

3.2.4. Características, modificadores y complementos

Las características (*features*) son una vía que se utiliza para manipular la oración, para conseguir el resultado deseado, como por ejemplo los tiempos verbales. Por defecto, la oración recibe una serie de características como puede ser el tiempo verbal presente de indicativo. Si el objetivo es una sentencia en tiempo verbal pasado, es necesario indicar ese cambio. Esto se consigue mediante el uso de las características.
Por ejemplo, utilizando la sentencia "*My father builds the house.*" como base, en pasado será "*My father built the house.*". Eso se consigue añadiendo el siguiente código antes del realizador:

```
p.setFeature(Feature.TENSE, TENSE.PAST);
```

Lo que devuelve es la frase deseada:

```
My father built the house.
```

Si el objetivo a comunicar es la sentencia "*My father does not build the house*", en lugar del código anterior, hay que usar la negación:

```
p.setFeature(Feature.NEGATED, true);
```

El resultado va a ser:

```
My father does not build the house.
```

Las características TENSE, NEGATED, y INTERROGATIVE_TYPE son ejemplos de funciones que se pueden establecer en un *SPhraseSpec*. Hay muchas más características como MODAL, PASSIVE, PERFECT, y PROGRESSIVE. La información detallada sobre las características admitidas se puede consultar en la documentación API de *SimpleNLG*.

Los modificadores (*modifiers*) permiten añadir descripción a los elementos de la sentencia. Por ejemplo, si lo que se quiere es añadir el adjetivo "*smart*" al sujeto y crear la sentencia "*My smart father builds the house*", se puede hacer utilizando los modificadores. Sin embargo, es preciso separar las palabras que forman la sentencia en correspondientes tipos:

```
NPPhraseSpec subject = nlgFactory.createNounPhrase("My father");
VPPhraseSpec verb = nlgFactory.createVerbPhrase("build");
NPPhraseSpec object = nlgFactory.createNounPhrase("the house");
```

Ahora se puede añadir el modificador "*smart*" al objeto *subject*:

```
subject.addModifier("smart");
```

Se asignan los objetos a la estructura *SPhraseSpec* definida anteriormente:

```
p.setSubject(subject);
p.setObject(object);
p.setVerb(verb);
String output3 = realiser.realiseSentence(p);
System.out.println(output3);
```

El resultado será:

```
My smart father builds the house.
```

De manera similar podemos asignar un adverbio al verbo:

```
verb.addModifier("quickly");
```

Obteniendo de resultado la sentencia:

```
My smart father quickly builds the house.
```

Los complementos (*complements*) de *SimpleNLG* son cualquier elemento que aparece después del verbo. Si se ha especificado un objeto, el complemento aparecerá después de este. Algunos ejemplos de complementos:

- María es *feliz*.
- María escribió la carta *rápidamente*.
- María se dió cuenta *de que sus vacaciones han terminado*.

Para *SimpleNLG* todas las palabras que están en cursiva de las frases anteriores son complementos aunque tengan diferentes funciones dentro de la oración; desde el punto de vista de *SimpleNLG* todos son complementos y aparecen después del verbo.

Aunque *SimpleNLG* tiene una comprensión de los sujetos, verbos, y objetos, Éste tiene un concepto muy limitado de sintagmas adjetivales, sintagmas adverbiales, clausulas "que" o de otros elementos que pueden aparecer después del verbo. Pero si entiende el concepto de complemento, y por lo tanto es capaz de generar frases de ese tipo.

Los complementos se añaden al objeto tipo *SPhraseSpec*, por ejemplo:

```
p.addComplement("very quickly");
```

Usando el ejemplo base, se obtiene:

```
      My smart father builds the house very quickly.
```

3.3. Componentes principales de SimpleNLG

En los apartados anteriores se describe la creación y manipulación básica de sentencias en *SimpleNLG*. Sin embargo, sus capacidades son mucho mayores. La interfaz API completa está disponible en la página de los autores.

SimpleNLG está compuesto de varios módulos. Cada módulo se encuentra en un paquete Java constituido por las clases que lo implementan.

Los principales módulos son (*ver figura 3. Princpales módulos de SimpleNLG*):

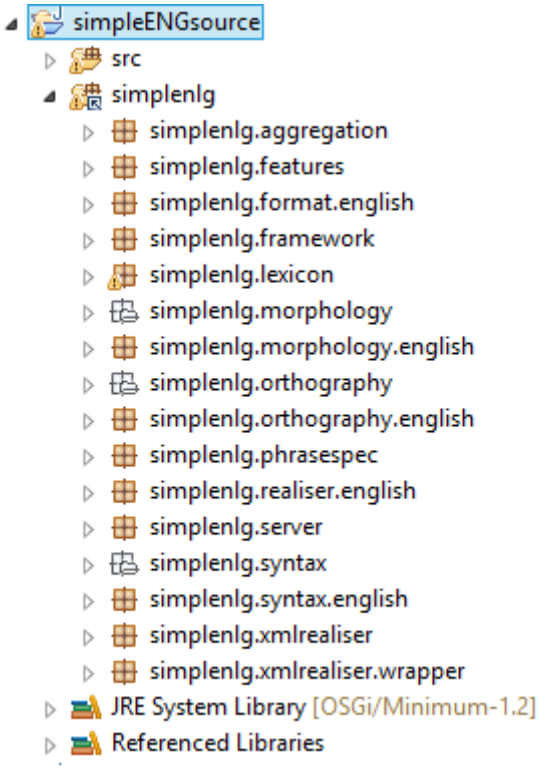

Figura 3. Principales módulos de SimpleNLG.

- **simplenlg.aggregation:** Constituido por nueve clases que manipulan la agregación de sentencias.
- **simplenlg.features:** En este módulo se encuentran las características que se utilizan para modificar la frase, por ejemplo el número, el género, los tiempos verbales, entre otros. Son tipos enumerados.
- **simplenlg.format.english:** Este módulo añade una gestión del documento por ejemplo añadir un título, nombres de los capítulos, viñetas, entre otros.
- **simplenlg.framework:** Este paquete contiene los componentes básicos necesarios para ejecutar *SimpleNLG*, por ejemplo *NLGFactory*, *NLGElement*, entre otros.

- **simplenlg.lexicon:** Contiene clases que permiten cargar y manipular el lexicón. También contiene el fichero XML con el lexicón básico inglés.
- **simplenlg.morphology.english:** Este paquete contiene las clases necesarias para ejecutar el procesador de la morfología del idioma Inglés.
- **simplenlg.orthography.english:** El paquete contiene sólo una clase con reglas las de ortografía del idioma inglés, como comenzar la frase con la letra mayúscula o añadir punto al final.
- **simplenlg.phrasespec**: Módulo con las clases que definen los tipos de sentencia como *VPPhraseSpec*, *SPPhraseSpec*, entre otros.
- **simplenlg.syntax.english:** Este paquete contiene las clases necesarias para ejecutar el procesador de sintaxis para el idioma Inglés.

Capítulo 4

ADAPTACIÓN DE SIMPLENLG AL IDIOMA ESPAÑOL

4. Adaptación de SimpleNLG al idioma Español

En este apartado se presentan las modificaciones realizadas para desarrollar un prototipo de *SimpleNLG* en el idioma Español. Los primeros pasos consisten en el estudio preliminar de la herramienta, para comprender su funcionamiento interno, hacer las primeras pruebas de implementación y posteriormente seleccionar un conjunto de reglas de gramática y sintaxis española candidatas para la implementación del prototipo.

4.1. SimpleNLG versión plurilingüe (ingles-francés).

Durante el estudio previo de la herramienta, se han encontrado (referenciados en la página oficial de *SimpleNLG*) enlaces a versiones bilingües (ingles/francés, e inglés/alemán en desarrollo).

La versión de francés no sólo estaba modificada para aceptar el idioma francés e inglés, sino que ya presentaba un carácter plurilingüe.

Esta versión desarrollada por Pierre-Luc Vaudry, [Vaudry, 2011] de la Universidad de Montreal presenta las siguientes características:

- Genera sentencias en francés gramáticamente correctas.
- Pocas modificaciones de la herramienta respecto a la versión original:
    - Mantiene la misma API
    - Motor interno casi sin alterar
- Generación bilingüe.
- Facilidad de adición de otros idiomas.

Esta versión, resultó ser la candidata perfecta para la inclusión de la implementación correspondiente al idioma Español, de ésta manera se conseguía en una sóla librería, ofrecer cobertura para los tres idiomas.

Además se reduce la posibilidad de redundar en el código - volver a crear las interfaces y todas las modificaciones necesarias para darle el carácter plurilingüe. Se ha decidido partir de este trabajo para comenzar la implementación del idioma Español.

En esta versión de *SimpleNLG* se ha modificado la estructura de los paquetes anteriormente vista. Se han extraído los elementos generales, dejándolos en los paquetes genéricos. Las clases que dependían del idioma se han movido a un módulo correspondiente, creando interfaces Java para múltiples implementaciones.

Algunas de las modificaciones que ha tenido que hacer el autor para adaptar la versión básica de la herramienta al idioma francés han sido:

- Adaptación de la sintaxis:
    - Sintagma nominal (colocación de determinante, concordancia con el adjetivo y determinante).
    - Sintagma verbal (negación, participio).
- Adaptación de la morfología:
    - Género femenino, el número plural.

    - Tiempos verbales (algunos).
    - Pronombres personales.
- Añadir un módulo nuevo para tratamiento de la morfo-fonología[2]:
    - Preposiciones (ejemplo: “de + le → au”)
    - Elision (le + homme → l'homme) y Liaison (le + beau + homme→ le bel homme)
- Cambios en la estructura de SimpleNLG:
    - Cada NLGElement “conoce” su idioma.
- Lexicón de idioma francés:
    - 3871 palabras.

El trabajo realizado es inmenso. La modificación, aunque todavía no cubre toda la gramática francesa (según el autor), está preparada para aceptar otros idiomas, característica que se aprovecha este Trabajo Fin de Máster.

4.2. Implementación

4.2.1. Análisis

El objetivo principal de ésta fase de análisis fue entender los cambios introducidos, las tareas que han de realizar los nuevos módulos, y sus relaciones entre sí, todo esto con la finalidad de diseñar una solución para la adaptación de la herramienta al idioma español.

La metodología que se ha seguido, es la considerada ingeniería inversa[3], ya que no existe documentación sobre el diseño interno de la herramienta, es por ello que el primer paso en el proceso de análisis fue incorporar la librería *SimpleNLG* del idioma francés a un nuevo proyecto de Eclipse y estudiar detenidamente cada clase de cada módulo.

Tras el análisis de esta nueva arquitectura de *SimpleNLG*, se ha identificado que los cambios introducidos han separado los módulos correspondientes de cada idioma, de los módulos globales que presentan características compartidas o son interfaces que son necesarias implementar para cada idioma. Por lo tanto, la incorporación del idioma Español va a seguir esta misma arquitectura.

En la figura 4, se representa la composición de los módulos de *SimpleNLG* de la versión francesa.

El análisis ha permitido obtener la siguiente información detallada sobre el funcionamiento del *SimpleNLG* bilingue.

- La parte del *framework* que define la jerarquía de clases que cubren las unidades léxicas, frases y elementos del documento, tales como párrafos, pueden ser mantenidos en común para los diferentes idiomas. Algunas normas y principios gramaticales compartidos están en clases abstractas de las cuales los módulos específicos del idioma pueden derivar.
  Muchos de los métodos estáticos del idioma inglés se modificaron a métodos regulares con el fin de poder sobrescribirlos en las nuevas subclases.

[2] La morfo-fonología se define como la relación entre la morfología y la fonética.
[3] La ingeniería inversa tiene como objetivo obtener información o un diseño del producto, con el fin de determinar cómo está hecho, que lo hace funcionar y como ha sido fabricado.

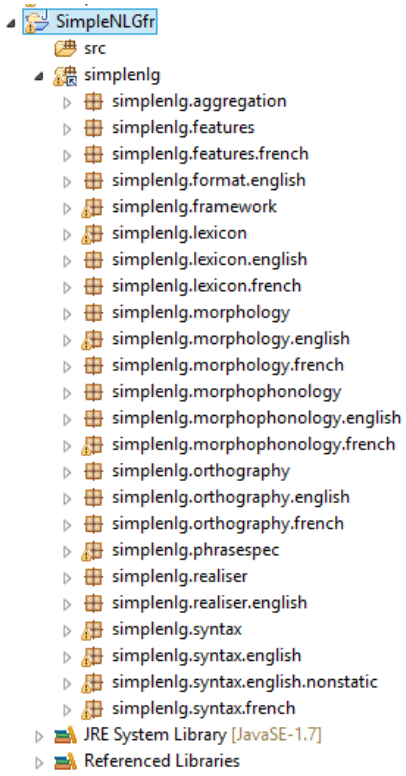

Figura 4. Estructura de paquetes de la versión francesa de SimpleNLG.

- Características: *SimpleNLG* utiliza un sistema de características para diversas funciones:
    - Codificar propiedades morfológicas y sintácticas de las unidades léxicas.
    - Permitir al usuario establecer parámetros de una frase en particular (singular/plural, tiempos verbales, entre otros).
    - Mantener el control (internamente) sobre el contenido de la frase y diversa información necesaria durante la realización.

  Este sistema es suficientemente genérico para ser utilizado para otros idiomas. La mayoría de las funciones son reutilizables y otras pueden ser añadidas según sea necesario.

- Lexicón: en *SimpleNLG*, el lexicón ya está relativamente bien separado de la gramática.
  La clase básica de lexicón proporciona una interfaz a un archivo XML que contiene la información necesaria acerca de las unidades léxicas. La lista de campos disponibles en éste archivo puede ser fácilmente extendida mediante la adición de características léxicas a las utilizadas para el inglés. Va a ser necesario incluir nuevas características para tener en cuenta la mayor complejidad de la morfología española.
- Sintaxis: en primer lugar, los idiomas inglés, francés y español tienen los mismos constituyentes básicos de la oración en el mismo orden: sujeto-verbo-objeto (SVO). Como este orden es relativamente estable, libera, en la mayoría de los casos, de tener que elegir entre distintos órdenes de palabras sintácticamente correctas.
- Sintagma nominal: Los tres idiomas pueden tener un determinante en el comienzo de una frase. Además, en *SimpleNLG*, un sintagma nominal puede tener pre y post modificadores. Los adjetivos, en inglés, por defecto, son considerados pre modificadores y todo lo demás post modificadores.

Sin embargo, esto no es tan claro para los otros idiomas; en francés y español los adjetivos pueden ser colocados después del sustantivo. Esto se ha solucionado añadiendo nueva característica léxica.

Además, en estos idiomas el determinante y los adjetivos deben concordar con el sustantivo en género y número. En lugar de añadir un nuevo mecanismo para propagar características donde se necesiten, la solución implementada fue dejar que el determinante y los adjetivos obtengan la información necesaria de su componente principal (sustantivo). De esta manera la gestión es más flexible facilitando la realización multilingüe.

- Sintagma preposicional: se utilizan preposiciones para introducir diversos complementos en todos los idiomas.
- Tiempos verbales: las formas verbales tanto del francés como del español son más variadas que las del inglés.
- El nivel de morfo-fonología es un nuevo nivel de procesamiento. Ha sido añadido debido a la necesidad de analizar dos palabras adyacentes y no introducirlo en el módulo de morfología, ya que éste opera sobre una palabra en el momento.
- La generación bilingüe implementada es capaz de determinar dinámicamente el idioma de cada unidad de procesamiento: las sentencias para el módulo de sintaxis, las unidades léxicas para el módulo de morfología, entre otras.

  Las factorías utilizadas por el usuario para crear las especificaciones de las estructuras sintácticas o crear unidades léxicas, utilizan el lexicón específico de cada idioma. Entonces, cada módulo de procesamiento elige en el momento de realización qué conjunto de reglas aplicar a un procesamiento determinado basándose en el idioma del lexicón. Por lo tanto, las oraciones, frases y palabras de diferentes idiomas se pueden mezclar libremente.
- Aunque se pueden añadir más idiomas, tal vez sería mejor especificar en común un formalismo gramatical, en lugar de insertarlo en los módulos de procesamiento. Sin embargo, esto requeriría un cambio drástico de la arquitectura.

4.2.2. Diseño

Para poder empezar con la implementación de los módulos correspondientes al idioma español es necesario identificar todos los elementos que van a ser necesarios. En esta fase se ha diseñado la adaptación de la herramienta *SimpleNLG* al idioma español.

El diseño mantiene la misma estructura de diseño que siguen los demas idiomas. De esta manera se evitan modificaciones en el motor interno de la herramienta o en la API por lo que el funcionamiento de los otros idiomas no se ve afectado.

Los módulos del idioma español se diseñaron de manera que sean capaces de poder cubrir la totalidad de la gramática española, es decir, no va a ser necesario añadir nuevos módulos posteriormente. Sin embargo, la limitación del tiempo y la extensión de la gramática española no hacen posible la implementación completa en este Trabajo Fin de Master. Es por ello que se ha seleccionado un cierto conjunto de reglas gramaticales que son las más representativas y que permiten generar la mayor cantidad de lenguaje bien formado.

Los módulos que han sido implementados para el idioma español son los siguientes:

Módulo de características: Este módulo contiene las definiciones de las características especiales para el idioma español.

- La clase *Tense* contiene los enumerados que indican el tipo de tiempo verbal. Esta clase es universal para todos los idiomas, pero esta modificación no influye en el funcionamiento de estos. Los tiempos verbales que es capaz conjugar el programa son:
    - Indicativo
        - Presente
        - Futuro simple
        - Pretérito perfecto compuesto
        - Pretérito imperfecto
        - Pretérito pluscuamperfecto
        - Pretérito perfecto simple
        - Pretérito anterior
        - Futuro perfecto
        - Condicional simple
        - Condicional perfecto
    - Subjuntivo
        - Presente
        - Pretérito perfecto
        - Pretérito pluscuamperfecto
        - Futuro simple
- La clase *SpanishLexicalFeature* contiene variables tipo *String* para relacionar todos los tiempos verbales con las personas; por ejemplo: *presente + 1 + s = presente1s*, lo que significa, que el tiempo verbal es presente, primera persona del singular.
  Estas variables se utilizan en el lexicón para marcar la conjugación irregular de un verbo. El procesador morfológico elige la conjugación correspondiente según estas variables.

Módulo de morfología: este módulo se ocupa de las conjugaciones y modificaciones de los diferentes tipos de palabras como los determinantes, sustantivos, adjetivos, verbos, adverbios y pronombres. En la Figura 5, se puede ver el diseño de este módulo.

La interfaz que se mantiene genérica para los diferentes idiomas define los métodos que deberían aparecer en la clase que la implemente. Cada de estos métodos se ocupa de procesamiento de diferente tipo de palabra:

- *doNounMorphology(InflectedWordElement, WordElement):* Trata los sustantivos. *InflectedWordElement* contiene la palabra a declinar mientras en *WordElement* guarda la palabra sin conjugar.
  El método devuelve un *StringElement* que contiene la palabra modificada. Éstas clases son utilizadas y no simples *Strings* porque es necesario mantener las diferentes características de cada palabra (qué palabra es, qué persona, número, entre otros) para poder hacer las modificaciones correctas. En este método se trata si el sustantivo es masculino o femenino y singular o plural.

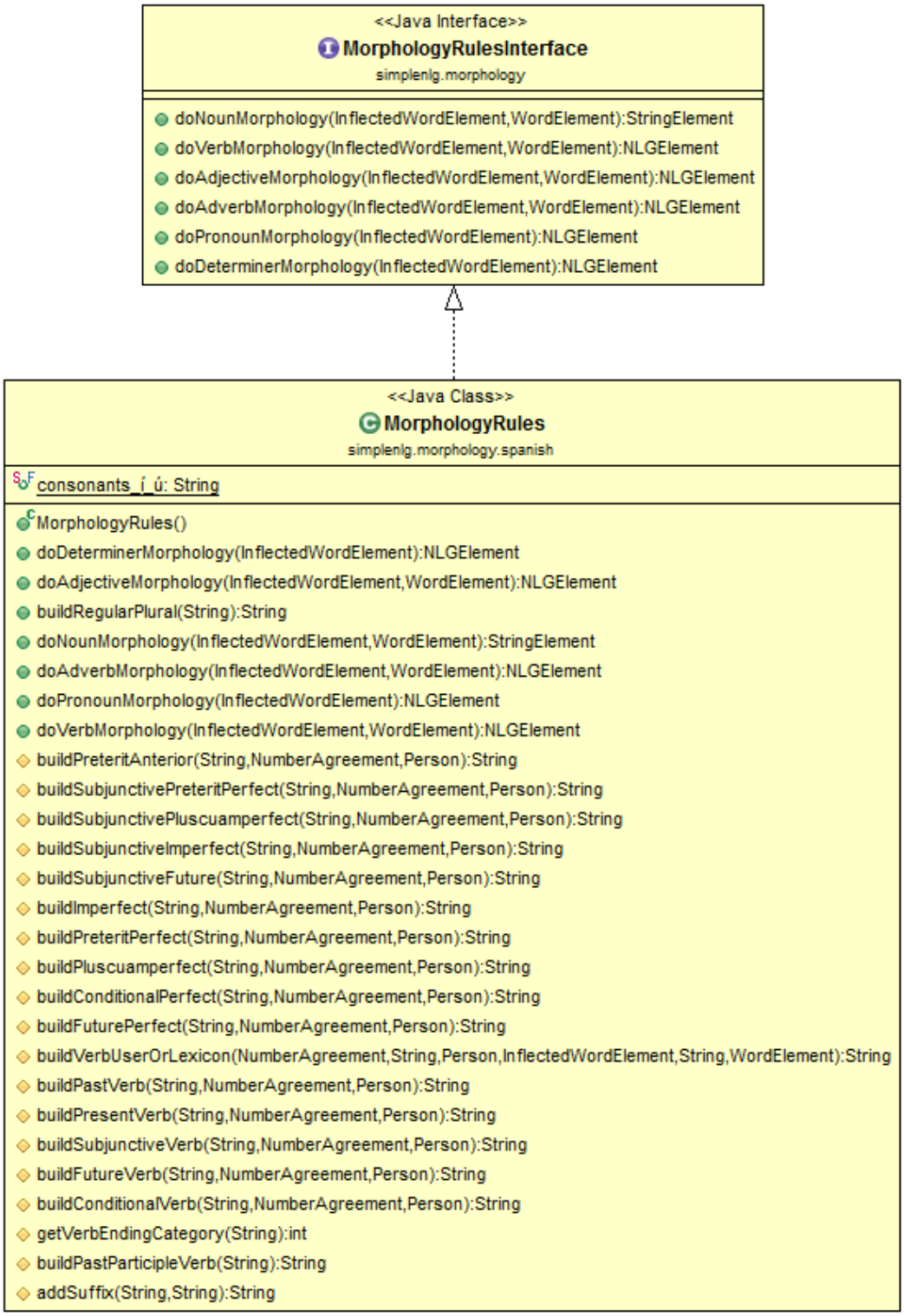

Figura 5. Diagrama UML del módulo de morfología

- *doVerbMorphology(InflectedWordElement, WordElement):* Trata los verbos. Acepta los mismos argumentos que el método anterior. Desde este método se invocan los métodos que hacen la conjugación del verbo (los métodos *buildAnyTense()*) según las características establecidas (el tiempo verbal), o según las características internas como el número o la persona.

- *doAdjectiveMorphology(InflectedWordElement, WordElement):* Trata los adjetivos. Aplica las modificaciones necesarias para que el adjetivo concuerde con el sustantivo, por lo que accede a las características de éste exactamente a número y género.
- *doAdverbMorphology(InflectedWordElement, WordElement):* Los adverbios en su gran mayoría están tratados en la parte de sintaxis, pero en la parte de morfología, es decir, este método, se recuperan las excepciones del lexicón.
- *doPronunMorphology(InflectedWordElement, WordElement):* Trata los pronombres. Este método no está implementado en el prototipo actual.
- *doDeterminerMorphology(InflectedWordElement, WordElement):* Trata los determinantes. Opera con los mismos elementos que los métodos anteriores, pero con valores de un determinante (un, una, unos, unas, el, la, los, las). La tarea de este método es concordar el determinante con el sustantivo al que pertenece en persona y número, por lo tanto accede a las características de éste.
- *buildAnyTense(String, NumberAgreement, Person):* Estos métodos implementan los tiempos verbales del español. Los argumentos que reciben son: *String* es el verbo en forma básica a conjugar, *NumberAgreement* es el número (plural, singular) en el que se quiere que este el verbo conjugado y *Person* es la persona (primera, segunda y tercera). Invoca *getVerbEndingCategory()* para obtener la categoría del verbo y *addSuffix()* para concatenar el lexema con el sufijo de la conjugación.
- *getVerbEndingCategory(String):* Este método es el que se encarga de revisar la terminación del verbo. En caso del español "ar", "er" e "ir". Devuelve la categoría al que pertenece el verbo; primera, segunda o tercera.

Módulo de morfo-fonología: Este módulo introducido en la versión francesa también es necesario en la versión española, ya que también hay que tratar la situación cuando aparece una preposición "de" seguida de un determinante "el" o la preposición "a" seguida del determinante "el". En la Figura 6, se puede ver el diseño de este módulo.

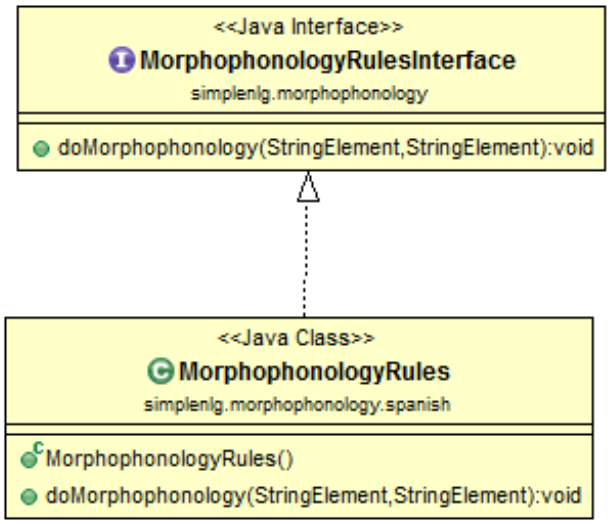

Figura 6. Diagrama UML del módulo de mofo-fonología

La interfaz también está compartida por clases de otros idiomas.

- *doMorphophonology(StringElement, StringElement):* Este método, recibe dos *StringElement*. Su tarea consiste, si es necesario, en eliminar el determinante de la oración, y transformarlo en una preposición de tipo "del" o "al". La modificación la hace directamente sobre el primer objeto *StringElement*.

Módulo de lexicón: Este módulo contiene una clase Java que se ocupa de cargar el fichero XML del lexicón y leer las palabras de éste. El fichero de lexicón contiene palabras del diccionario español en una estructura de etiquetas que facilitan la lectura de las diferentes declinaciones o excepciones.

Para este trabajo se han incluido un total de 36 palabras en forma básica, que se han utilizado para realizar las pruebas. También aparecen las conjugaciones de algunos de los verbos irregulares.

Las etiquetas que se pueden utilizar están contenidas en el módulo de características. El siguiente fragmento es un ejemplo de una entrada en el fichero:

```
<word>
      <base>yo</base>
      <category>pronoun</category>
      <pronoun_type>personal</pronoun_type>
      <person>first</person>
      <number>singular</number>
      <discourse_function>subject</discourse_function>
</word>
```

Éste ejemplo pertenece a la entrada de la palabra "yo", es un pronombre personal, persona primera de singular y su función en la oración es de sujeto.

Para cada tipo de palabra es necesario el uso de otras etiquetas, como en el caso del verbo irregular "saber" su registro en el fichero es el siguiente:

```
<word>
      <base>saber</base>
      <category>verb</category>
      <pastParticiple>sabido</pastParticiple>
      <gerund>sabiendo</gerund>
      <present1s>sé</present1s>
      <past1s>supe</past1s>
      <past2s>supiste</past2s>
      <past3s>supo</past3s>
      <past1p>supimos</past1p>
      <past2p>supisteis</past2p>
      <past3p>supieron</past3p>
      <future1s>sabré</future1s>
      <future2s>sabrás</future2s>
      <future3s>sabrá</future3s>
      <future1p>sabremos</future1p>
      <future2p>sabréis</future2p>
      <future3p>sabrán</future3p>
      <conditional1s>sabría</conditional1s>
      <conditional2s>sabrías</conditional2s>
      <conditional3s>sabría</conditional3s>
      …
</word>
```

No se incluye la entrada completa del verbo por motivos de espacio, pero es suficiente este fragmento de código para entender la estructura del lexicón. Las diferentes etiquetas identifican la conjugación del verbo a un tiempo verbal determinado.

Es importante destacar que sólo se incluyen los casos de excepción y no toda la conjugación del verbo en todos los tiempos verbales, ya que de ello se ocupa el módulo de morfología.

Módulo de ortografía: El módulo de ortografía se ha mantenido el mismo que en otros idiomas. Aunque ha sido necesario añadir una pequeña modificación, el cambio de la preposición ("y") que se usa para relacionar las oraciones coordinadas.

Módulo de sintaxis: Es el módulo más extenso y completo. Sin embargo, la implementación del idioma francés de este módulo es relativamente válida para el idioma español. Por lo tanto, la versión de este módulo para el idioma español se basa en la versión francesa con varias modificaciones que sobretodo incluyen los cambios en las palabras que se añaden, por ejemplo para formar la negación ("no").

Aunque, se ha podido reutilizar este módulo para el idioma español, la mayoría de la gramática que no se ha podido cubrir tiene que estar implementada en las clases contenidas en este módulo; el modo pasivo, la interrogación o las palabras clíticas, son algunos de los ejemplos que necesitan revisión.

4.2.3. Implementación

Tras las fases de análisis y diseño de la adaptación, se procedió con la fase de implementación que ha consistido principalmente en:

- Añadir el idioma español al conjunto de idiomas de *SimpleNLG* y referenciarlo en las clases genéricas.
- Crear paquetes correspondientes a los módulos del idioma español.
- Crear las clases con los métodos para implementar.
- Añadir las nuevas características correspondientes a los tiempos verbales del español.
- Implementar los métodos.

Además, también se han llevado a cabo otras tareas requeridas para poder conseguir la modificación:

- Preparar una clase de test con el método Main.
- Llevar el proceso de test mediante "prueba-error" y posteriormente el proceso validación.

Los algoritmos que se desarrollaron, se encuentran principalmente en el módulo de morfología. Para explicar el procesamiento que hacen algunos de los métodos de la clase *MorphologyRules* se describen los algoritmos en pseudocódigo.

El método *doVerbMorphology()* es el método que se ocupa de invocar el método de conjugación de tiempo verbal correspondiente, para ello lee las características de la palabra. Sin embargo, antes invoca a un método que revisa primero si para este verbo, tiempo verbal, persona y número hay una entrada en el lexicón.

```
doVerbMorphology(element, baseWord){

	numero=element.leerCaracteristica(Caracteristica.Numero)
	persona=element.leerCaracteristica(Caracteristica.Persona)
	tiempoVerbal=element.leerCaracteristica(Caracteristica.TiempoVerbal)

	si tiempoVerbal==TiempoVerbal.Presente entonces
		verbo=recuperarDeLexicon(numero, tiempoVerbal, persona, element, baseWord)
		si verbo==null entonces
			verbo=conjugaPresente(element, numero, persona)
	y si tiempoVerbal==TiempoVerbal.Pasado entonces
		verbo=recuperarDeLexicon(numero, tiempoVerbal, persona, element, baseWord)
		si verbo==null entonces
			verbo=conjugaPasado(element, numero, persona)
	y si tiempoVerbal==TiempoVerbal.Futoro entonces
		verbo=recuperarDeLexicon(numero, tiempoVerbal, persona, element, baseWord)
		si verbo==null entonces
		verbo=conjugaFutro(element, numero, persona)
	y si tiempoVerbal==TiempoVerbal.Imperfecto entonces
		verbo=recuperarDeLexicon(numero, tiempoVerbal, persona, element, baseWord)
		si verbo==null entonces
		verbo=conjugaImperfecto(element, numero, persona)
	…
	return verbo
}
```

Los métodos que hacen directamente la construcción del verbo conjugado, que están en el módulo de morfología, siguen el siguiente algoritmo:

```
 buildTiempoVerbal(element, number, person){
	categoría=obtenerCategoriaVerbo(element)
	lexema=element.substring(0, element.length-2)

	si number==SINGULAR entonces
		si categoría==1 entonces
			si person==primera entonces
				sufjio=sufjio_primera_persona_singular_tiempo_verbal_catAR
			si person==segunda entonces
				sufjio=sufjio_segunda_persona_singular_tiempo_verbal_catAR
			si person==tercera entonces
				sufjio=sufjio_tercera_persona_singular_tiempo_verbal_catAR
		si categoría==2 entonces
			si person==primera entonces
				sufjio=sufjio_primera_persona_singular_tiempo_verbal_catER
			si person==segunda entonces
				sufjio=sufjio_segunda_persona_singular_tiempo_verbal_catER
			si person==tercera entonces
				sufjio=sufjio_tercera_persona_singular_tiempo_verbal_catER
		si categoría==3 entonces
			si person==primera entonces
				sufjio=sufjio_primera_persona_singular_tiempo_verbal_catIR
			si person==segunda entonces
				sufjio=sufjio_segunda_persona_singular_tiempo_verbal_catIR
			si person==tercera entonces
```

```
                        sufjio=sufjio_tercera_persona_singular_tiempo_verbal_catIR
      si number==PLURAL entonces
            si categoría==1 entonces
                  si person==primera entonces
                        sufjio=sufjio_primera_persona_plural_tiempo_verbal_catAR
                  si person==segunda entonces
                        sufjio=sufjio_segunda_persona_plural_tiempo_verbal_catAR
                  si person==tercera entonces
                        sufjio=sufjio_tercera_persona_plural_tiempo_verbal_catAR
            si categoría==2 entonces
                  si person==primera entonces
                        sufjio=sufjio_primera_persona_plural_tiempo_verbal_catER
                  si person==segunda entonces
                        sufjio=sufjio_segunda_persona_plural_tiempo_verbal_catER
                  si person==tercera entonces
                        sufjio=sufjio_tercera_persona_plural_tiempo_verbal_catER
            si categoría==3 entonces
                  si person==primera entonces
                        sufjio=sufjio_primera_persona_plural_tiempo_verbal_catIR
                  si person==segunda entonces
                        sufjio=sufjio_segunda_persona_plural_tiempo_verbal_catIR
                  si person==tercera entonces
                        sufjio=sufjio_tercera_persona_plural_tiempo_verbal_catIR
      return concatena(lexema+sufjio)
}
```

Este algoritmo elige el sufijo correspondiente al tiempo verbal, persona, número y categoría del verbo y posteriormente lo concatena con el lexema.

Otro ejemplo del algoritmo que utiliza el método *doDeterminerMorphology(InflectedWordElement element).*

```
doDeterminerMorphology(element1, element2) {

      padre=element.obtenerPadre()
      genero=padre.obtenerCaracteristica(Caracteristica.Genero)
      femenino=(Genero.Femenino==genero)
      inflectedForm=element

      si element==PLURAL entonces
            inflectedForm = leerCaracteristica(Caracteristica.Plural)
            si femenino entonces
                  inflectedForm== leerCaracteristica(Caracteristica.Plural_Femenino)
      y si femenino && element==SINGULAR entonces
            inflectedForm== leerCaracteristica(Caracteristica.Singular_Femenino)

      return inflectedForm
}
```

Este método trata la morfología del determinante. Modifica el objeto *element* sólo si el género del padre es femenino o número plural.

En el módulo de morfo-fonología también hay un algoritmo en el método que se ocupa de eliminar el determinante “el” cuándo es precedido de las preposiciones “de” o “a”.

```
doMorphophonology(leftWord, rightWord) {

	categoriaIzquierda=leftWord.obtenerCagetoria()
	categoriaDerecha=rightWord.obtenerCagetoria()

	si categoriaIzquierda == Preposicion && categoriaDerecha == Determinante entonces
		si leftWord == “de” && rightWord == “el” entonces
			leftWord.valor(“del”) && rightWord.valor(null)
		si leftWord == “a” && rightWord == “el” entonces
			leftWord.valor(“al”) && rightWord.valor(null)
}
```

La estructura final de la herramienta *SimpleNLG* adaptado al idioma español se puede observar en la siguiente figura 6.

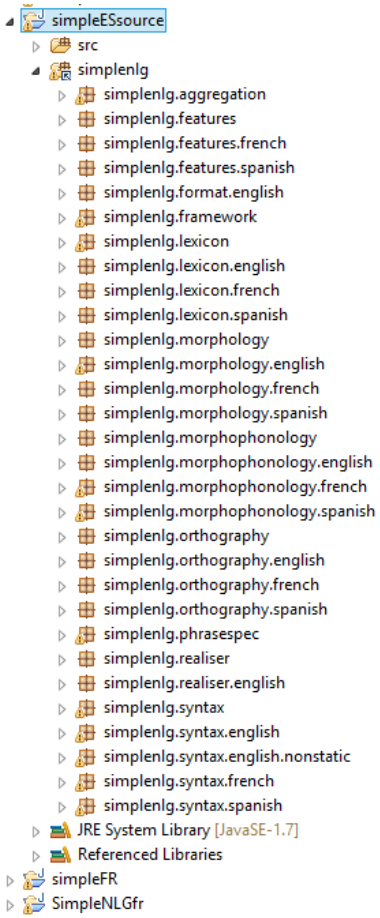

Figura 7. Estructura del prototipo de *SimpleNLG* español.

4.3. Ejemplo de la creación de una sentencia en español

La versión adaptada al español presenta la misma API que la versión francesa, por lo tanto la creación de las sentencias es la misma.

A continuación, en la figura 7, se muestra un sencillo ejemplo del proceso:

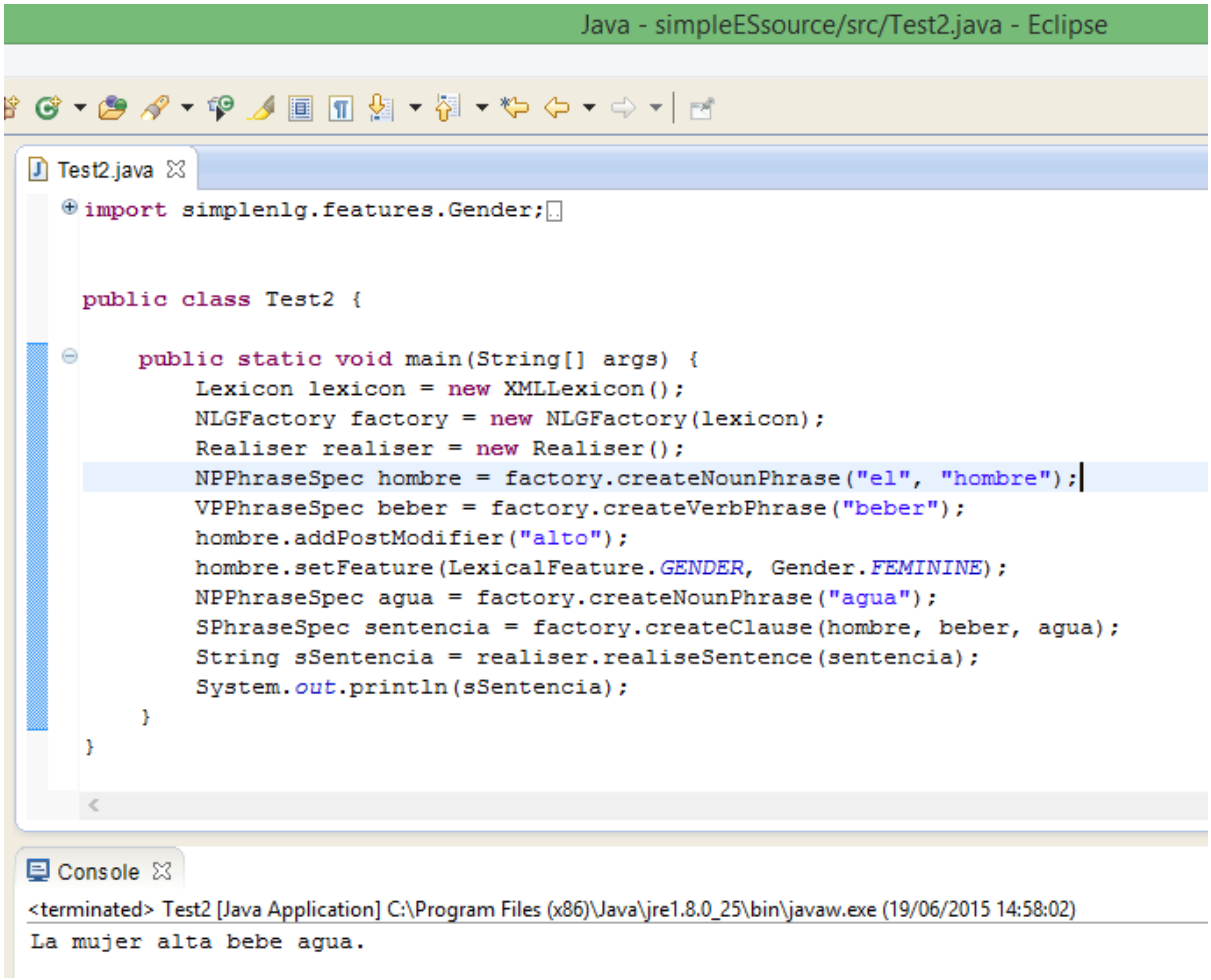

Figura 8. Ejemplo de la creación de una sentencia en español.

Al igual que en los otros ejemplos es necesario inicializar los objetos que permiten el manejo de la estructura de la oración; el lexicón, que se pasa como argumento al constructor del *NLGFactory* y el realizador.
Se definen los diferentes elementos de la sentencia, a cada uno se asignan las modificaciones requeridas y se genera la sentencia usando el *NLGFactory* y guardando el objeto resultando en un objeto tipo *SPhraseSpec*. Ahora, el realizador recibe esta estructura lógica para devolver una cadena de texto. En este caso se imprime por consola pero podría utilizare una entrada a otro programa.

4.4. Conjunto de reglas gramaticales y sintácticas cubiertas

La Gramática Española es muy extensa, motivo por el cual, el recurso utilizado para consultarla es el siguiente: "Nueva gramática de la lengua española" [RAE, 2009-11], el cual tiene más de mil páginas en su versión resumida.

Por lo tanto, tratar toda la gramática no ha sido una tarea simple de conseguir. Se ha seleccionado un conjunto de reglas que se consideran de más común uso, que son las que se listan a continuación:

- Conjugación de los verbos regulares, en todos los tiempos verbales. Las conjugaciones de verbos irregulares tienen que aparecer en el lexicón del cual se elige la expresión correcta según el tiempo, número y la persona. Las terminaciones de verbos regulares que se han considerado son:
    - "ar", por ejemplo comprar, cocinar, etc.
    - "er", por ejemplo beber, correr, etc.
    - "ir", por ejemplo cumplir, partir, etc.
- Declinación de los sustantivos según el número y el género.
- Declinación de los adjetivos según el número y el género del "padre" (sustantivos al que "describe").
- Negación.
- Agregación de frases con el mismo sujeto, por ejemplo: El perro ladra. El perro corre. El perro come. Resulta en una frase: El perro ladra, corre y come. Hay que destacar la inclusión de la coma y la preposición "y" antes del último verbo.
- La composición de la preposición "de" o "a" procedida del artículo "el", formando la preposición "del" o "al" respectivamente.

Además, hay que indicar que la sintaxis en mayor medida está cubierta por la versión básica de *SimpleNLG* como puede ser por ejemplo el comienzo de la frase con una letra mayúscula y terminación con un punto.

Capítulo 5

PRUEBAS Y VALIDACIÓN

5. Pruebas y Validación

Se han hecho numerosas pruebas durante la fase de desarrollo, finalmente, cuando la funcionalidad se consideraba cubierta, se creaba un caso de prueba final.

En las siguientes tablas aparecen los casos de prueba que se han hecho. Se han incluido también casos que no están cubiertos por el programa en la versión actual para dar información sobre las posibles líneas de futuro trabajo. Las características por defecto son: género masculino, número singular, tiempo verbal presente.

El objetivo de esta validación es hacer pruebas que cubran toda la gramática que se ha pretendido incluir en el prototipo. Se considera un caso de prueba cuando la herramienta hace la realización de una sentencia. Las pruebas más exhaustivas se han podido hacer sobre la conjugación de los verbos en todos los tiempos verbales del español.

Prueba 1: Tiempos Verbales

Estas pruebas incluyen conjugación de verbos en español. Se ha utilizado tres verbos regulares para cubrir la conjugación de las tres categorías de terminación; “ar”, “er” e “ir”. Estos verbos son "comprar", "beber" y "permitir". Para el caso de los verbos irregulares también se ha utilizado tres verbos "andar", "saber" e "huir". Siendo su conjugación irregular, los casos se obtienen del lexicón por lo que también fue necesario incluirlos en la validación

En los casos de prueba no se pretendía construir textos muy elaborados sino revisar si el programa efectúa la conjugación correctamente. Para hacer estas pruebas se ha preparado un programa de test que recorre tres listas: lista de los verbos, lista de los tiempos verbales y la lista de las personas. Los resultados se muestran en la *Tabla 2. Tiempos verbales*, un total de 540 casos de pruebas

		TIEMPOS VERBALES															
		INDICATIVO									SUBJUNTIVO						
	PRONOMBRES	PRESENTE	PASADO	FUTURO	IMPERFECTO	CONDICIONAL	CONDICIONAL PERFECTO	PRETÉRITO PERFECTO	PLUSCUAMPERFECTO	FUTURO PERFECTO	FUTURO	IMPERFECTO	PLUSCUAMPERFECTO	PRETÉRITO PERFECTO	PRESENTE	PRETÉRITO ANTERIOR
Comprar	Yo	compro	compré	compraré	compraba	compraría	habría comprado	he comprado	hubiera comprado	habré comprado	comprare	comprara	hubiera comprado	haya comprado	compre	hube comprado
	Tu	compras	compraste	comprarás	comprabas	comprarías	habrías comprado	has comprado	hubieras comprado	habrás comprado	comprares	compraras	hubieras comprado	hayas comprado	compres	hubiste comprado
	Él/Ella	compra	compró	comprará	compraba	compraría	habría comprado	ha comprado	hubiera comprado	habrá comprado	comprare	comprara	hubiera comprado	haya comprado	compre	hubo comprado
	Nosotros	compramos	compramos	compraremos	comprábamos	compraríamos	habríamos comprado	hemos comprado	hubiéramos comprado	habremos comprado	compráremos	compráramos	hubiéramos comprado	hayamos comprado	compremos	hubimos comprado
	Vosotros	comprais	comprasteis	compraréis	comprabais	compraríais	habríais comprado	habéis comprado	hubierais comprado	habréis comprado	comprareis	comprarais	hubierais comprado	hayáis comprado	compreis	hubisteis comprado
	Ellos/Ellas	compran	compraron	comprarán	compraban	comprarían	habrían comprado	han comprado	hubieran comprado	habrán comprado	compraren	compraran	hubieran comprado	hayan comprado	compren	hubieron comprado
Beber	Yo	bebo	bebí	beberé	bebía	bebería	habría bebido	he bebido	hubiera bebido	habré bebido	bebiere	bebiera	hubiera bebido	haya bebido	beba	hube bebido
	Tu	bebes	bebiste	beberás	bebías	beberías	habrías bebido	has bebido	hubieras bebido	habrás bebido	bebieres	bebieras	hubieras bebido	hayas bebido	bebas	hubiste bebido
	Él/Ella	bebe	bebió	beberá	bebía	bebería	habría bebido	ha bebido	hubiera bebido	habrá bebido	bebiere	bebiera	hubiera bebido	haya bebido	beba	hubo bebido
	Nosotros	bebemos	bebimos	beberemos	bebíamos	beberíamos	habríamos bebido	hemos bebido	hubiéramos bebido	habremos bebido	bebiéremos	bebiéramos	hubiéramos bebido	hayamos bebido	bebamos	hubimos bebido
	Vosotros	bebeis	bebisteis	beberéis	bebíais	beberíais	habríais bebido	habéis bebido	hubierais bebido	habréis bebido	bebiereis	bebierais	hubierais bebido	hayáis bebido	bebais	hubisteis bebido
	Ellos/Ellas	beben	bebieron	beberán	bebían	beberían	habrían bebido	han bebido	hubieran bebido	habrán bebido	bebieren	bebieran	hubieran bebido	hayan bebido	beban	hubieron bebido
Permitir	Yo	permito	permití	permitiré	permitía	permitiría	habría permitido	he permitido	hubiera permitido	habré permitido	permitiere	permitiera	hubiera permitido	haya permitido	permita	hube permitido
	Tu	permites	permitiste	permitirás	permitías	permitirías	habrías permitido	has permitido	hubieras permitido	habrás permitido	permitieres	permitieras	hubieras permitido	hayas permitido	permitas	hubiste permitido
	Él/Ella	permite	permitió	permitirá	permitía	permitiría	habría permitido	ha permitido	hubiera permitido	habrá permitido	permitiere	permitiera	hubiera permitido	haya permitido	permita	hubo permitido
	Nosotros	permitimos	permitimos	permitiremos	permitíamos	permitiríamos	habríamos permitido	hemos permitido	hubiéramos permitido	habremos permitido	permitiéremos	permitiéramos	hubiéramos permitido	hayamos permitido	permitamos	hubimos permitido
	Vosotros	permitis	permitisteis	permitiréis	permitíais	permitiríais	habríais permitido	habéis permitido	hubierais permitido	habréis permitido	permitiereis	permitierais	hubierais permitido	hayáis permitido	permitais	hubisteis permitido
	Ellos/Ellas	permiten	permitieron	permitirán	permitían	permitirían	habrían permitido	han permitido	hubieran permitido	habrán permitido	permitieren	permitieran	hubieran permitido	hayan permitido	permitan	hubieron permitido
Andar	Yo	ando	anduve	andaré	andaba	andaría	habría andado	he andado	hubiera andado	habré andado	andare	andara	hubiera andado	haya andado	ande	hube andado
	Tu	andas	anduviste	andarás	andabas	andarías	habrías andado	has andado	hubieras andado	habrás andado	andares	andaras	hubieras andado	hayas andado	andes	hubiste andado
	Él/Ella	anda	anduvo	andará	andaba	andaría	habría andado	ha andado	hubiera andado	habrá andado	andare	andara	hubiera andado	haya andado	ande	hubo andado
	Nosotros	andamos	anduvimos	andaremos	andábamos	andaríamos	habríamos andado	hemos andado	hubiéramos andado	habremos andado	andáremos	andáramos	hubiéramos andado	hayamos andado	andemos	hubimos andado
	Vosotros	andais	anduvisteis	andaréis	andabais	andaríais	habríais andado	habéis andado	hubierais andado	habréis andado	andareis	andarais	hubierais andado	hayáis andado	andeis	hubisteis andado
	Ellos/Ellas	andan	anduvieron	andarán	andaban	andarían	habrían andado	han andado	hubieran andado	habrán andado	andaren	andaran	hubieran andado	hayan andado	anden	hubieron andado
Saber	Yo	sé	supe	sabré	sabía	sabería	habría sabido	he sabido	hubiera sabido	habré sabido	sabiere	sabiera	hubiera sabido	haya sabido	sepa	hube sabido
	Tu	sabes	supiste	sabrás	sabías	saberías	habrías sabido	has sabido	hubieras sabido	habrás sabido	sabieres	sabieras	hubieras sabido	hayas sabido	sepas	hubiste sabido
	Él/Ella	sabe	supo	sabrá	sabía	sabería	habría sabido	ha sabido	hubiera sabido	habrá sabido	sabiere	sabiera	hubiera sabido	haya sabido	sepa	hubo sabido
	Nosotros	sabemos	supimos	sabremos	sabíamos	sabríamos	habríamos sabido	hemos sabido	hubiéramos sabido	habremos sabido	sabiéremos	sabiéramos	hubiéramos sabido	hayamos sabido	sepamos	hubimos sabido
	Vosotros	sabeis	supisteis	sabréis	sabíais	sabríais	habríais sabido	habéis sabido	hubierais sabido	habréis sabido	sabiereis	sabierais	hubierais sabido	hayáis sabido	sepáis	hubisteis sabido
	Ellos/Ellas	saben	supieron	sabrán	sabían	sabrían	habrían sabido	han sabido	hubieran sabido	habrán sabido	sabieren	sabieran	hubieran sabido	hayan sabido	sepan	hubieron sabido
Huir	Yo	huyo	huí	huiré	huía	huiría	habría huido	he huido	hubiera huido	habré huido	huiere	huiera	hubiera huido	haya huido	huya	hube huido
	Tu	huyes	huiste	huirás	huías	huirías	habrías huido	has huido	hubieras huido	habrás huido	huieres	huieras	hubieras huido	hayas huido	huyas	hubiste huido
	Él/Ella	huye	huyó	huirá	huía	huiría	habría huido	ha huido	hubiera huido	habrá huido	huiere	huiera	hubiera huido	haya huido	huya	hubo huido
	Nosotros	huimos	huimos	huiremos	huíamos	huiríamos	habríamos huido	hemos huido	hubiéramos huido	habremos huido	huiéremos	huiéramos	hubiéramos huido	hayamos huido	huyamos	hubimos huido
	Vosotros	huis	huisteis	huiréis	huíais	huiríais	habríais huido	habéis huido	hubierais huido	habréis huido	huiereis	huierais	hubierais huido	hayáis huido	huyamos	hubisteis huido
	Ellos/Ellas	huyen	huyeron	huirán	huían	huirían	habrían huido	han huido	hubieran huido	habrán huido	huieren	huieran	hubieran huido	hayan huido	huyan	hubieron huido

Tabla 2. Tiempos verbales.

Prueba 2: Complementos, características y modificadores

En los siguientes casos de prueba se testea añadir complementos, preposiciones, negación, cambiar género, numero o agregar diferentes frases con el mismo sujeto.

Caso de prueba	Entrada	Características	Resultado Esperado	Resultado Obtenido
541	Sujeto: maría Predicado: hablar Complemento: juan		María habla con Juan.	María habla con Juan.
542	Sujeto: maría Predicado: correr Preposición: en Complemento: el parque		María corre en el parque.	María corre en el parque.
543	Sujeto: juan Predicado: escuchar Complemento: música	Negación	Juan no escucha música.	Juan no escucha música.
544	Sujeto: el hombre Predicado: beber Complemento: agua	Género: femenino	La mujer bebe agua.	La mujer bebe agua.
545	Sujeto: el chico Predicado: escuchar Complemento: música	Numero: Plural	Los chicos escuchan música.	Los chicos escuchan música.
546	Sujeto: la mujer Adjetivo: alto Predicado: beber Complemento: agua		La mujer alta bebe agua.	La mujer alta bebe agua.
547	Sujeto: Juan. Predicado: beber Predicado: comer Predicado: correr	Coordinación de frases	Juan come, bebe y corre.	Juan come, bebe y corre.

Tabla 3.Complementos, características y modificadores.

Prueba 3: Concatenación de preposiciones

Las pruebas de concatenación de preposiciones “de” y “el”, y “a” y “el”.

Caso de prueba	Entrada	Características	Resultado Esperado	Resultado Obtenido
548	Sujeto: el hombre Predicado: huir Preposición: de Complemento: el perro		Los chicos huyen del perro.	Los chicos huyen del perro.
549	Sujeto: maría Predicado: besar Preposición: a Complemento: el hombre		María besa al hombre alto.	María besa al hombre alto.

Tabla 4. Concatenación de preposiciones.

Prueba 4: Funcionalidad incompleta

En la *Tabla 5. Funcionalidad incompleta* se han incluido casos los cuales el prototipo no es capaz de cubrir completamente como los participios con función de adjetivos. Se puede observar que los dos primeros casos de prueba se hacen correctamente pero al cambiar el género y el número no se modifican adecuadamente.

Caso de prueba	Entrada	Características	Resultado Esperado	Resultado Obtenido
550	Sujeto: el ángel Complemento: caer	Participio	El ángel caído.	El ángel caído.
551	Sujeto: el toro Complemento: perder	Participio	El toro perdido.	El toro perdido.
552	Sujeto: el toro Complemento: perder	Participio Género: Femenino	El toro perdida.	La vaca perdido.
553	Sujeto: el toro Complemento: perder	Participio Numero: Plural	Los toros perdidos.	Los toros perdido.

Tabla 5. Funcionalidad incompleta

Capítulo 6

CONCLUSIONES

6. Conclusiones

La realización del presente Trabajo de Fin de Máster fue un reto para el autor y, sin ninguna duda, una vía de aprendizaje de algo nuevo. El objetivo ideal, "Conseguir una adaptación de la herramienta *SimpleNLG* al idioma español, en los inicios del trabajo fue considerado como una meta lejana, casi inalcanzable. Durante la fase de Estudio del Arte, fueron adquiridos los conocimientos sobre la temática de la Generación de Lenguaje Natural, usando diferentes fuentes de información pero sobretodo, basándose en la publicación de la profesora Maria Bernardos Socorro "¿Qué es la generación de lenguaje natural? Una visión general sobre el proceso de generación"".

Las tareas involucradas en el proceso de GLN, no son tareas completamente separadas con fronteras rigurosamente definidas, más bien, son conjunto de acciones que tienen que ocurrir para la GLN correcta que se pueden cubrir en menor o mayor medida según las necesidades o los objetivos comunicativos del texto.

Para completar el estudio de estado de arte, fue necesario conocer herramientas de Generación de Lenguaje Natural. El aprendizaje de las diferentes "maneras" de generación de lenguaje, revisar arquitecturas y funciones de los generadores otorgó un conocimiento adicional que permitió entender mejor el objetivo de este Trabajo y centrar la atención del autor en el desarrollo de la modificación.

La propia herramienta *SimpleNLG*, presenta una arquitectura relativamente sencilla. La API disponible no es muy extensa pero permite diferentes maneras de creación de estructuras léxicas basándose sobre todo en tipo de la palabra tratada. La realización, convierte las estructuras lógicas al texto, consultando en primera instancia la configuración del usuario, luego el lexicón y finalmente, haciendo uso de las reglas de morfología y sintaxis si los anteriores no dan resultado.

Una vez entendido el proceso de generación que sigue el *SimpleNLG* se ha procedido a desarrollar la modificación. Para ello, se ha utilizado una versión del *SimpleNLG* bilingüe inglés-francés. Se han incorporado los módulos necesarios para la generación de textos en español, añadiendo la morfología española de las palabras, reglas de la gramática y de sintaxis.

Teniendo cubierto un conjunto significativo de la Gramática Española, se ha procedido a la validación y pruebas de lo desarrollado. Se han diseñado 553 casos de prueba incluyendo algunos casos erróneos para demostrar las funcionalidades sin cubrir. Tal y como ya se ha mencionado no es una validación exhaustiva pero es suficiente para demostrar la funcionalidad correcta de la modificación.

Concluyendo, se considera superado el objetivo ideal - creación de la extensión a la herramienta *SimpleNLG* y los objetivos intermedios necesarios como: análisis de GLN actual, estudio del periodismo de datos como una aplicación de GLN, estado del arte de diferentes herramientas de GLN, análisis detallado de la herramienta *SimpleNLG* en su versión básica y finalmente el desarrollo de la modificación y su validación.

.

Capítulo 7

LÍNEAS FUTURAS DE TRABAJO

7. Líneas Futuras de Trabajo

Aunque la versión actual de la librería, cubre gran conjunto de las reglas gramaticales del español y de su sintaxis, todavía queda mucho trabajo para completar. Las futuras líneas de trabajo podrían entonces ser:

- **Construcción de preguntas:** Ya existe código que trata la situación de interrogación, pero no se tratan bien los diferentes tipos de preguntas; ¿Cómo?, ¿Qué?, ¿Dónde?, ¿Por qué?, entre otros.
- **Construcciones con gerundio:** También existe el código que hace posible tratar las construcciones de gerundio. Habría que añadir el verbo "estar" en tiempo verbal correspondiente.
- **Construcciones con participio como adjetivo:** También está desarrollado el código que permite tratar esa construcción, pero no lo hace del todo bien. Necesario revisar.
- **Lexicón:** En este Trabajo Fin de Master se ha utilizado un lexicón de ejemplo, donde se iban agregando palabras que se necesitaban para hacer las pruebas. Construir el lexicón que cubra gran número de palabras del español es una tarea que requiere mucho más tiempo y recursos adecuados, además no fue objetivo del este proyecto. Queda pendiente para incluso otro Trabajo Fin de Master.
- **Validación exhaustiva:** Incluir una validación mucho más extensa y exhaustiva que la que se ha utilizado en este Trabajo. Se han cubierto unos 30 casos de prueba. Se considera que una validación completa debería tener cerca de 2000 casos de prueba.

Capítulo 8

ANEXOS

8. Anexos

A. Manual de añadir el código fuente al proyecto en Eclipse.

Este manual explica paso a paso como incluir el código fuente de la librería *SimpleNLG* al proyecto.

1. Crear nuevo proyecto: Project (File>New>Project).
2. Añadir las siguientes librerias al "build path":
   - Project>Properties>Java Build Path.
   - Seleccionar la pestaña "Libraries".
   - Hacer click en el boton "Add external jars".
   - Buscar el fichero `simplenlg/lib` folder y seleccionar todos los jar.
   - Hacer click en el boton "Open".
   - Hacer click en el boton "OK".
3. Añadir el código Fuente de simpleNLG:
   - Abrir las propiedades del proyecto de nuevo y seleccionar la pestaña "Sources".
   - Hacer click en el botón "Link Source…" y añadir la ruta al fichero `simplenlg/src`, escribir `simplenlg` en el campo "folder name".
   - Hacer click en "Finish" y luego en "OK".
4. Modificar la codificación del texto
   - Properties > Resource > Text file encoding > Other
   - Seleccionar UTF-8 y luego hacer click en "OK"

B. Creación de la clase con el método Main y la importación de la librería simplenlg.

Crea una clase Java que contenga el método Main, por ejemplo "TestMain".

1. Incluye los siguientes imports:

```
import simplenlg.framework.*;
import simplenlg.lexicon.Lexicon;
import simplenlg.lexicon.spanish.XMLLexicon;
import simplenlg.realiser.*;
import simplenlg.phrasespec.*;
import simplenlg.features.*;
```

2. Añade el siguiente codigo:

```
public class TestMain {

        public static void main(String[] args) {
                Lexicon lexicon = Lexicon.getDefaultLexicon();
                NLGFactory nlgFactory = new NLGFactory(lexicon);
                Realiser realiser = new Realiser(lexicon);

        }

}
```

3. Para obtener un simple ejemplo:

```
NLGElement s1 = nlgFactory.createSentence("Mi perro es feliz");
String output = realiser.realiseSentence(s1);
System.out.println(output);
```

C. El fichero de pruebas

```
import java.util.Arrays;
import java.util.List;
import simplenlg.aggregation.ClauseCoordinationRule;
import simplenlg.features.Feature;
import simplenlg.features.Form;
import simplenlg.features.Gender;
import simplenlg.features.InterrogativeType;
import simplenlg.features.LexicalFeature;
import simplenlg.features.NumberAgreement;
import simplenlg.features.Tense;
import simplenlg.framework.*;
import simplenlg.lexicon.Lexicon;
import simplenlg.lexicon.spanish.XMLLexicon;
import simplenlg.realiser.*;
import simplenlg.phrasespec.*;

public class Test {

	public static void main(String[] args) {

		Lexicon lexicon = new XMLLexicon();
		NLGFactory factory = new NLGFactory(lexicon);
		Realiser realiser = new Realiser();

		// nombres
		NPPhraseSpec hombre = factory.createNounPhrase("el", "hombre");
		NPPhraseSpec chico = factory.createNounPhrase("el", "chico");
		NPPhraseSpec perro = factory.createNounPhrase("el", "perro");
		NPPhraseSpec maria = factory.createNounPhrase("Maria");
		NPPhraseSpec juan = factory.createNounPhrase("Juan");
		NPPhraseSpec agua = factory.createNounPhrase("agua");
		NPPhraseSpec musica = factory.createNounPhrase("música");
```

```
            NPPhraseSpec elParque = factory.createNounPhrase("el", "parque");
            NPPhraseSpec elAngel = factory.createNounPhrase("el", "ángel");
            NPPhraseSpec toro = factory.createNounPhrase("el", "toro");

            // verbos
            VPPhraseSpec beber = factory.createVerbPhrase("beber");
            VPPhraseSpec besar = factory.createVerbPhrase("besar");
            VPPhraseSpec correr = factory.createVerbPhrase("correr");
            VPPhraseSpec huir = factory.createVerbPhrase("huir");
            VPPhraseSpec hablar = factory.createVerbPhrase("hablar");
            VPPhraseSpec comer = factory.createVerbPhrase("comer");
            VPPhraseSpec escuchar = factory.createVerbPhrase("escuchar");
            VPPhraseSpec caer = factory.createVerbPhrase("caer");
            VPPhraseSpec perder = factory.createVerbPhrase("perder");

            // preposiciones
            PPPhraseSpec a = factory.createPrepositionPhrase("a");
            PPPhraseSpec con = factory.createPrepositionPhrase("con");
            PPPhraseSpec de = factory.createPrepositionPhrase("de");
            PhraseElement en = factory.createPrepositionPhrase("en");

            // Presente Simple 1º persona
            System.out.println("Prueba 1");
            SPhraseSpec presenteSimple1Pers = factory.createClause(maria, hablar);

            String sPresenteSimple1Pers =
realiser.realiseSentence(presenteSimple1Pers);
            System.out.println("\tPresente: \t" + sPresenteSimple1Pers);

            // Pasado Simple 1º persona
            SPhraseSpec pasadoSimple1Pers = factory.createClause(maria, hablar);
            hablar.setFeature(Feature.TENSE, Tense.PAST);
            String sPasadoSimple1Pers = realiser.realiseSentence(pasadoSimple1Pers);
            System.out.println("\tPasado: \t" + sPasadoSimple1Pers);

            // Futuro Simple 1º persona
```

```
SPhraseSpec futuroSimple1Pers = factory.createClause(maria, hablar);
hablar.setFeature(Feature.TENSE, Tense.FUTURE);
String sFuturoSimple1Pers = realiser.realiseSentence(futuroSimple1Pers);
System.out.println("\tFuturo: \t" + sFuturoSimple1Pers);

// Futuro perfecto
SPhraseSpec futuroPerf = factory.createClause(maria, hablar);
hablar.setFeature(Feature.TENSE, Tense.FUTURE_PERFECT);
String sFuturoPerf = realiser.realiseSentence(futuroPerf);
System.out.println("\tFuturo Perf: \t" + sFuturoPerf);

// Imperfecto
SPhraseSpec imperfecto = factory.createClause(maria, hablar);
hablar.setFeature(Feature.TENSE, Tense.IMPERFECT);
String sImperfecto = realiser.realiseSentence(imperfecto);
System.out.println("\tImperfecto: \t" + sImperfecto);

// Condicional
SPhraseSpec condicional = factory.createClause(maria, hablar);
hablar.setFeature(Feature.TENSE, Tense.CONDITIONAL);
String sCondicional = realiser.realiseSentence(condicional);
System.out.println("\tCondicional: \t" + sCondicional);

// Condicional perfecto
SPhraseSpec condPerfecto = factory.createClause(maria, hablar);
hablar.setFeature(Feature.TENSE, Tense.CONDITIONAL_PERFECT);
String sCondPerfecto = realiser.realiseSentence(condPerfecto);
System.out.println("\tCond Perfecto: \t" + sCondPerfecto);

// pluscuamperfecto
SPhraseSpec pluscuamperfecto = factory.createClause(maria, hablar);
hablar.setFeature(Feature.TENSE, Tense.PLUSCUAMPERFECT);
String sPluscuamperfecto = realiser.realiseSentence(pluscuamperfecto);
System.out.println("\tPluscuam: \t" + sPluscuamperfecto);

// preterito
```

```
            SPhraseSpec preterito = factory.createClause(maria, hablar);
            hablar.setFeature(Feature.TENSE, Tense.PRETERIT_PERFECT);
            String sPreterito = realiser.realiseSentence(preterito);
            System.out.println("\tPasado Perf: \t" + sPreterito);

            // Presente Simple 1º persona + preposición + nombre
            System.out.println("Prueba 2");
            SPhraseSpec presenteSimple1PersPrep = factory.createClause();
            presenteSimple1PersPrep.setSubject(maria);
            presenteSimple1PersPrep.setVerb(hablar);
            con.addComplement(juan);
            presenteSimple1PersPrep.addPostModifier(con);

            String sPresenteSimple1PersPrep =
realiser.realiseSentence(presenteSimple1PersPrep);
            System.out.println("\tPreposición: \t" + sPresenteSimple1PersPrep);

            // Presente Simple 1º persona + preposición + articulo + nombre
            SPhraseSpec presenteSimple1PersPrepArt = factory.createClause();
            presenteSimple1PersPrepArt.setSubject(maria);
            presenteSimple1PersPrepArt.setVerb(correr);
            en.addComplement(elParque);
            presenteSimple1PersPrepArt.addPostModifier(en);

            String sPresenteSimple1PersPrepArt =
realiser.realiseSentence(presenteSimple1PersPrepArt);
            System.out.println("\tPrep + art: \t" + sPresenteSimple1PersPrepArt);

            // Presente Simple + negacion
            System.out.println("Prueba 3");
            SPhraseSpec negacion = factory.createClause();
            negacion.setSubject(juan);
            negacion.setVerb(escuchar);
            negacion.setObject(musica);

            String sNegacionOri = realiser.realiseSentence(negacion);
            System.out.println("\tOriginal: \t" + sNegacionOri);
```

```
negacion.setFeature(Feature.NEGATED, true);

String sNegacion = realiser.realiseSentence(negacion);
System.out.println("\tNegada: \t" + sNegacion);

// cambio de genero
System.out.println("Prueba 4");
SPhraseSpec cambioGenero = factory.createClause();
cambioGenero.setSubject(hombre);
cambioGenero.setVerb(beber);
cambioGenero.setObject(agua);
String sCambioGenero = realiser.realiseSentence(cambioGenero);
System.out.println("\tOriginal: \t" + sCambioGenero);

hombre.setFeature(LexicalFeature.GENDER, Gender.FEMININE);
String sCambioGeneroFemenino = realiser.realiseSentence(cambioGenero);
System.out.println("\tFemenino: \t" + sCambioGeneroFemenino);

// singular y plural
System.out.println("Prueba 5");
SPhraseSpec singular = factory.createClause();
singular.setSubject(chico);
singular.setVerb(correr);
singular.setVerb(escuchar);
singular.setObject(musica);

String sSingular = realiser.realiseSentence(singular);
System.out.println("\tSingular: \t" + sSingular);

chico.setFeature(Feature.NUMBER, NumberAgreement.PLURAL);
String sPlural = realiser.realiseSentence(singular);
System.out.println("\tPlural: \t" + sPlural);

// adjetivo
System.out.println("Prueba 6");
```

```
SPhraseSpec adjetivo = factory.createClause();
adjetivo.setSubject(hombre);
hombre.addPostModifier("alto");
adjetivo.setVerb(beber);
adjetivo.setObject(agua);

String sAdjective = realiser.realiseSentence(adjetivo);
System.out.println("\tAdjetivo: \t" + sAdjective);

//agregación
System.out.println("Prueba 7");
SPhraseSpec frase1 = factory.createClause();
frase1.setSubject(juan);
frase1.setVerb(comer);
String sFrase1 = realiser.realiseSentence(frase1);
System.out.println("\tFrase 1: \t" + sFrase1);

SPhraseSpec frase2 = factory.createClause();
frase2.setSubject(juan);
frase2.setVerb(beber);
String sFrase2 = realiser.realiseSentence(frase2);
System.out.println("\tFrase 2: \t" + sFrase2);

SPhraseSpec frase3 = factory.createClause();
frase3.setSubject(juan);
frase3.setVerb(correr);
String sFrase3 = realiser.realiseSentence(frase3);
System.out.println("\tFrase 3: \t" + sFrase3);

ClauseCoordinationRule coord = new ClauseCoordinationRule();
List<NLGElement> elements = Arrays.asList((NLGElement) frase1,
            (NLGElement) frase2, (NLGElement) frase3);
List<NLGElement> result = coord.apply(elements);
NLGElement aggregated = result.get(0);
String sFrase4 = realiser.realiseSentence(aggregated);
System.out.println("\tAgregación: \t" + sFrase4);
```

```
//pasado irregular
System.out.println("Prueba 8");
SPhraseSpec irregular = factory.createClause(chico, huir);
huir.setFeature(Feature.TENSE, Tense.PAST);
String sIrregular = realiser.realiseSentence(irregular);
System.out.println("\tPasado Irregular: \t" + sIrregular);

//pasado irregular
SPhraseSpec presenteIrregular = factory.createClause(chico, huir);
huir.setFeature(Feature.TENSE, Tense.PRESENT);
String sPresenteIrregular = realiser.realiseSentence(presenteIrregular);
System.out.println("\tPresente Irregular: \t" + sPresenteIrregular);

//pasado irregular
SPhraseSpec futuroIrregular = factory.createClause(chico, huir);
huir.setFeature(Feature.TENSE, Tense.FUTURE);
String sFuturoIrregular = realiser.realiseSentence(futuroIrregular);
System.out.println("\tFuturo Irregular: \t" + sFuturoIrregular);

//verbo irregular + sustitución
System.out.println("Prueba 9");
huir.addPostModifier(de);
de.addComplement(perro);
huir.setFeature(Feature.TENSE, Tense.PRESENT);
SPhraseSpec sustitucionDel = factory.createClause(chico, huir);
String sSustitucionDel = realiser.realiseSentence(sustitucionDel);
System.out.println("\tSustitución de + el = del: \t" + sSustitucionDel);

besar.addPostModifier(a);
a.addComplement(hombre);
hombre.setFeature(LexicalFeature.GENDER, Gender.MASCULINE);
hombre.clearModifiers();
SPhraseSpec sustitucionAl = factory.createClause(maria, besar);
String sSustitucionAl = realiser.realiseSentence(sustitucionAl);
System.out.println("\tSustitución a + el = al: \t" + sSustitucionAl);
```

```
            // interrogación por qué
            System.out.println("Prueba 10");
            PhraseElement porque = factory.createClause("John", "comer");
            porque.addPostModifier("rápido");
            porque.setFeature(Feature.INTERROGATIVE_TYPE, InterrogativeType.YES_NO);
            String sPorque = realiser.realiseSentence(porque);
            System.out.println("\t¿Por qué?: \t" + sPorque);

            // gerundio
            System.out.println("Prueba 11");
            SPhraseSpec participio = factory.createClause();
            participio.setSubject(elAngel);
            elAngel.setComplement(caer);
            caer.setFeature(Feature.FORM, Form.PAST_PARTICIPLE);
            String sParticipio = realiser.realiseSentence(participio);
            System.out.println("\tParticipio: \t" + sParticipio);

            SPhraseSpec participioGenero = factory.createClause();
            participioGenero.setSubject(toro);
            toro.setComplement(perder);
            perder.setFeature(Feature.FORM, Form.PAST_PARTICIPLE);
            String sParticipioGenero = realiser.realiseSentence(participioGenero);
            System.out.println("\tOriginal: \t" + sParticipioGenero);

            toro.setFeature(LexicalFeature.GENDER, Gender.FEMININE);
            String sParticipioGeneroF = realiser.realiseSentence(participioGenero);
            System.err.println("\tFemenino: \t" + sParticipioGeneroF);

            toro.setFeature(LexicalFeature.GENDER, Gender.MASCULINE);
            toro.setFeature(Feature.NUMBER, NumberAgreement.PLURAL);
            String sParticipioGeneroPlural =
realiser.realiseSentence(participioGenero);
            System.err.println("\tPlural: \t" + sParticipioGeneroPlural);
      }
```

```
}
```

Referencias

[ACL, 2013] Association for Computational Linguistics. 2013. Downloadable NLG systems [Online]. Available: http://aclweb.org/aclwiki/index.php?title=Downloadable_NLG_systems

[Bernardos, 2007] Mª del Socorro Bernardos. "¿Qué es la generación de lenguaje natural? Una visión general sobre el proceso de generación" (2007) [Online]. Available: http://www.redalyc.org/articulo.oa?id=92503407

[Cahill y Reape, 1999] L. Cahill y M. Reape. 1999. "Component tasks in applied NLG systems" Technical Report ITRI-99-05. Information Technology Research Institute, University of Brighton (Reino Unido), 1999.

[Dixon, 1982] P. Dixon. "Plans and written directions for complex tasks". Journal of Verbal Learning and Verbal Behavior, 21:70-84, 1982.

[Krahmer et al., 2001] E. Krahmer, S. van Erk y A. Verleg. "A meta-algorithm for the generation of referring expressions". *Proceedings of the 8th European Workshop On Natural Language Generation* (ENLGW-2001). Toulouse (Francia), 6-7 julio 2001.

[RAE, 2009-11] Real Academia Española, 2009-20011."Nueva gramática de la lengua española".

[RAGS, 2000] Equipo del proyecto RAGS. "Towards a reference architecture for natural language generation systems". RAGS Technical Report. Information Technology Research Institute, University of Brighton (Reino Unido). Agosto de 2000. Última revisión: "The RAGS reference manual", 18 marzo 2002.

[Reiter y Dale, 1997] E. Reiter, R. Dale. "Building applied natural language generation systems". Journal of Natural Language Engineering, 3(1):57-87, 1997. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.8831&rep=rep1&type=pdf

[Reiter y Dale, 2000]. E. Reiter y R. Dale. "Building natural language generation systems". Cambridge University Press, Cambridge (Reino Unido), 2000.

[Gatt y Reiter, 2009] A. Gatt y E. Reiter. SimpleNLG: “A realisation engine for practical applications”. Department of Computing Science, University of Aberdeen (Reino Unido), 03 Junio 2009.2

[Vaudry, 2011] Pierre-Luc Vaudry. “Adding French to SimpleNLG” (2011) [Online].Available: http://www-etud.iro.umontreal.ca/~vaudrypl/snlgbil/ENLGpresentation.pdf

Recursos

Código SimpleNLG english Ehud Reiter, Albert Gatt, Dave Westwater, of Aberdeen University. “SimpleNLG”. [Online]. Available: https://code.google.com/p/simplenlg/

Código SimpleNLG english/french Pierre-Luc Vaudry. “SimpleNLG-EnFr 1.1,”. [Online]. Available: http://www-etud.iro.umontreal.ca/~vaudrypl/snlgbil/snlgEnFr_english.html